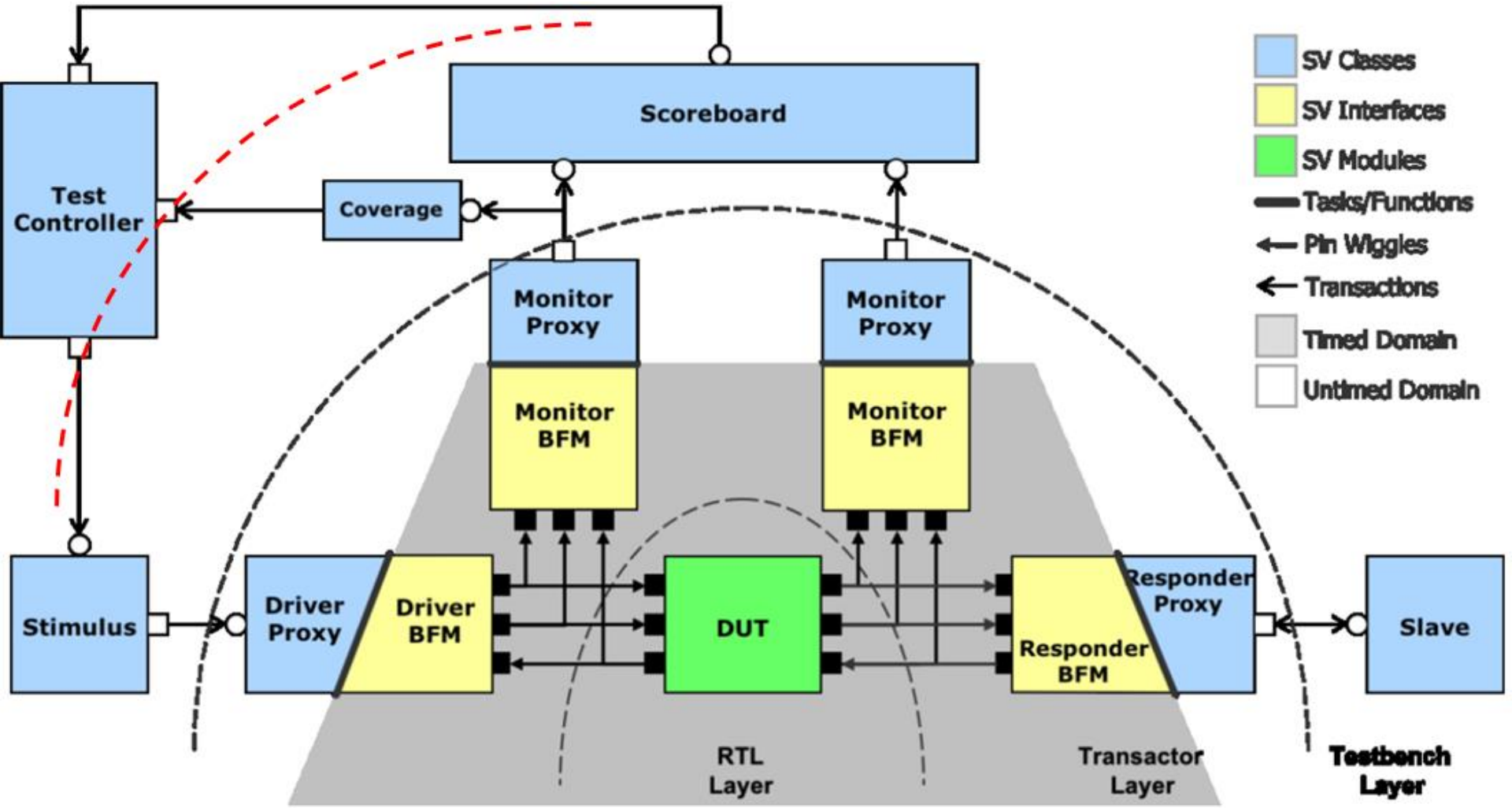
DUT and UVM Testbench
Multiple Agents

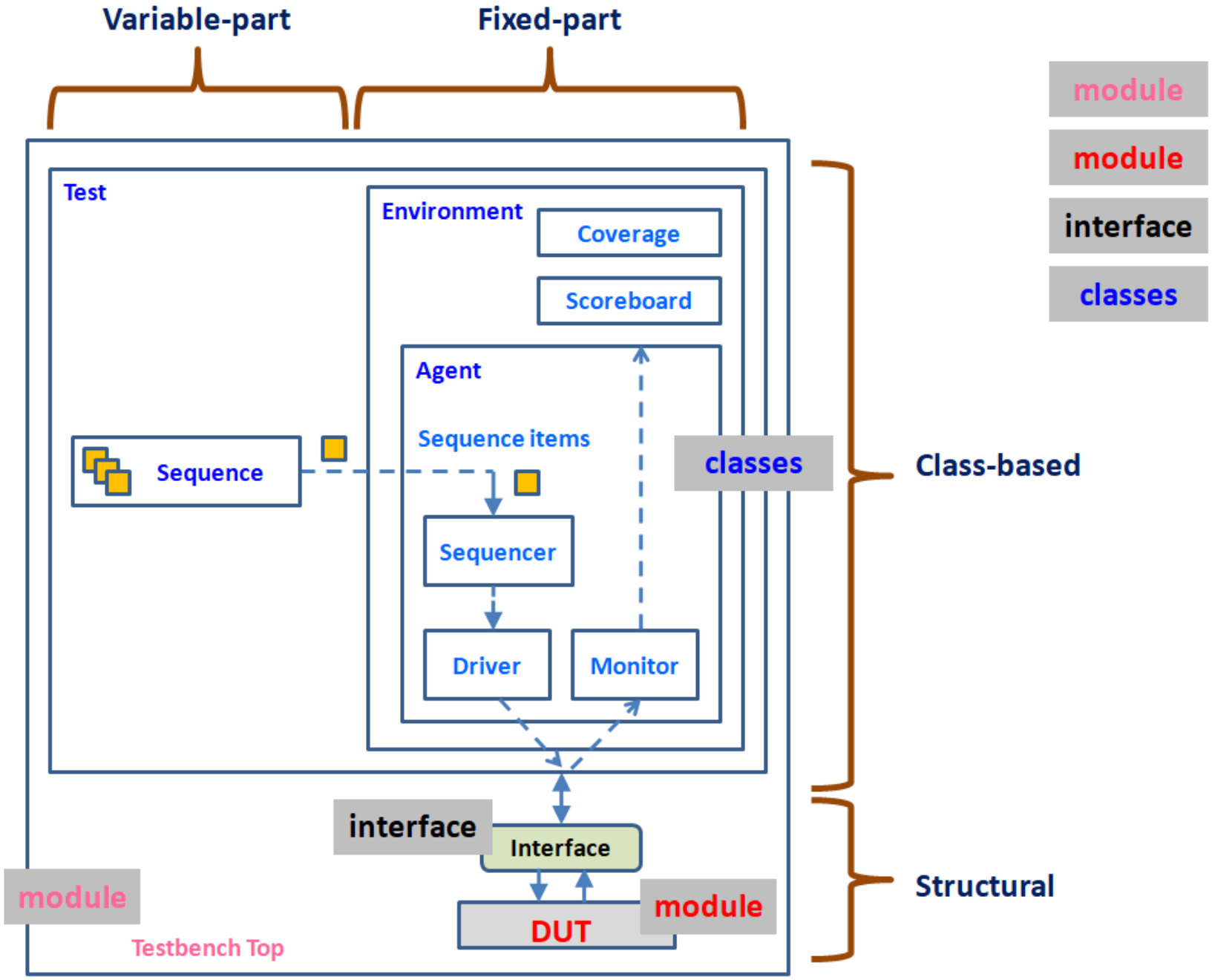
Tuan Nguyen-viet

Multiple Agents (cont'd)

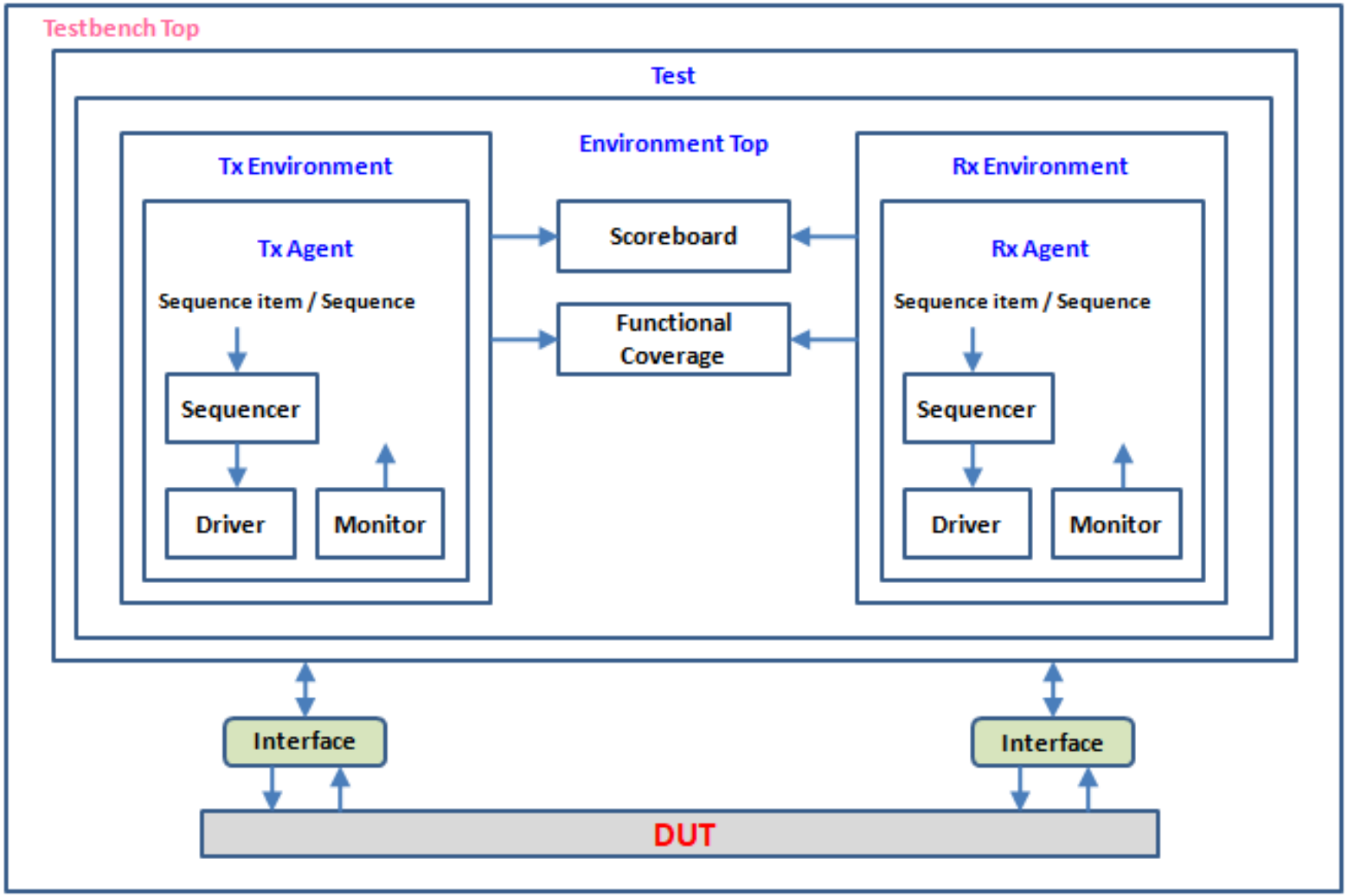
UVM Testbench Architecture (Cookbook)



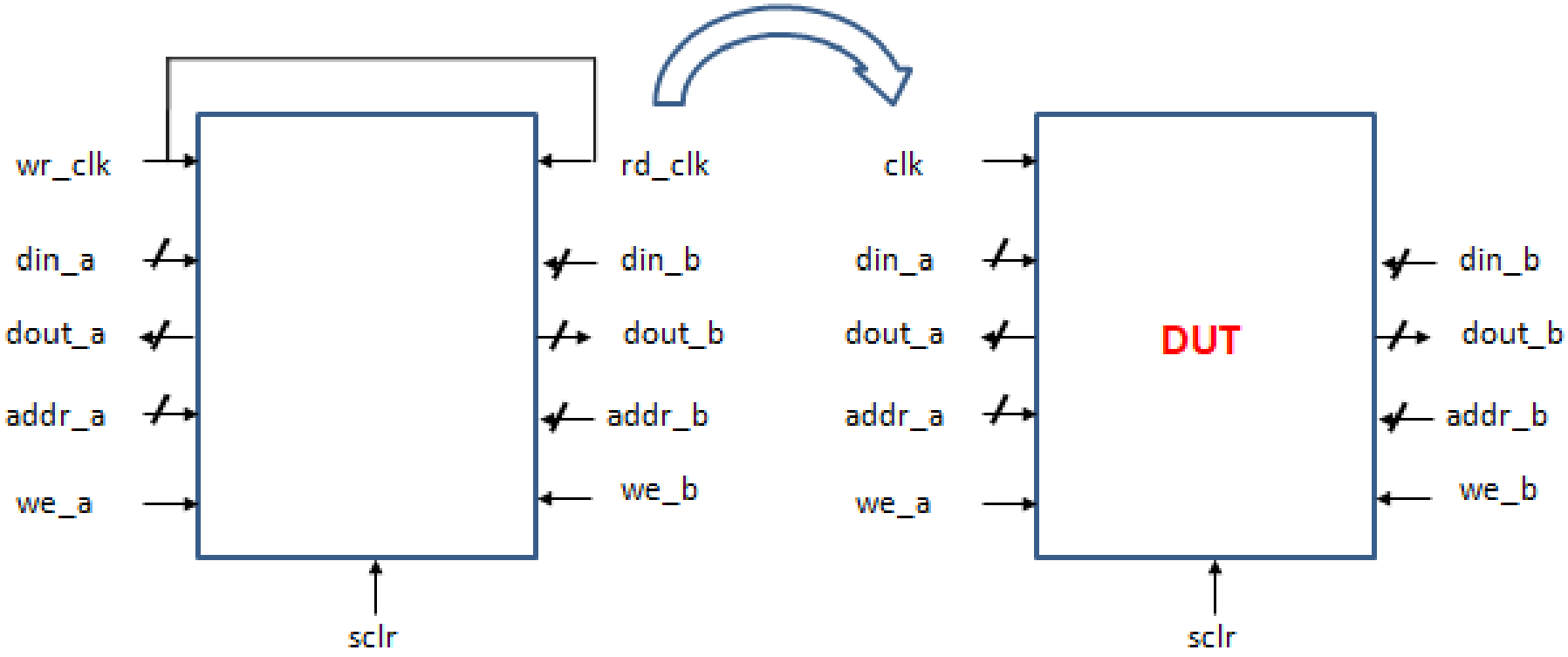
Bus functional model(BFM) is a model of physical interfaces of the DUT

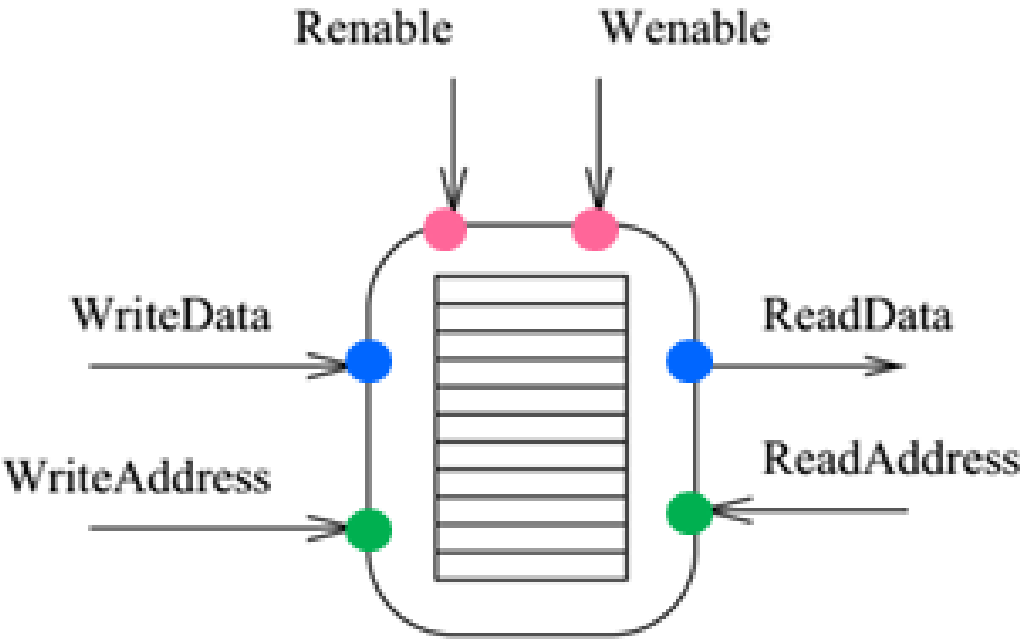

Variable-part
Fixed-part
module
module
interface
classes
Test
Environment
Coverage
Scoreboard
Agent
Sequence items
Sequence
classes
Class-based
Sequencer
Driver
Monitor
interface
Interface
module
DUT
module
Testbench Top
Structural

Key Components of a UVM Testbench



DUT – An Example: Dual Port RAM



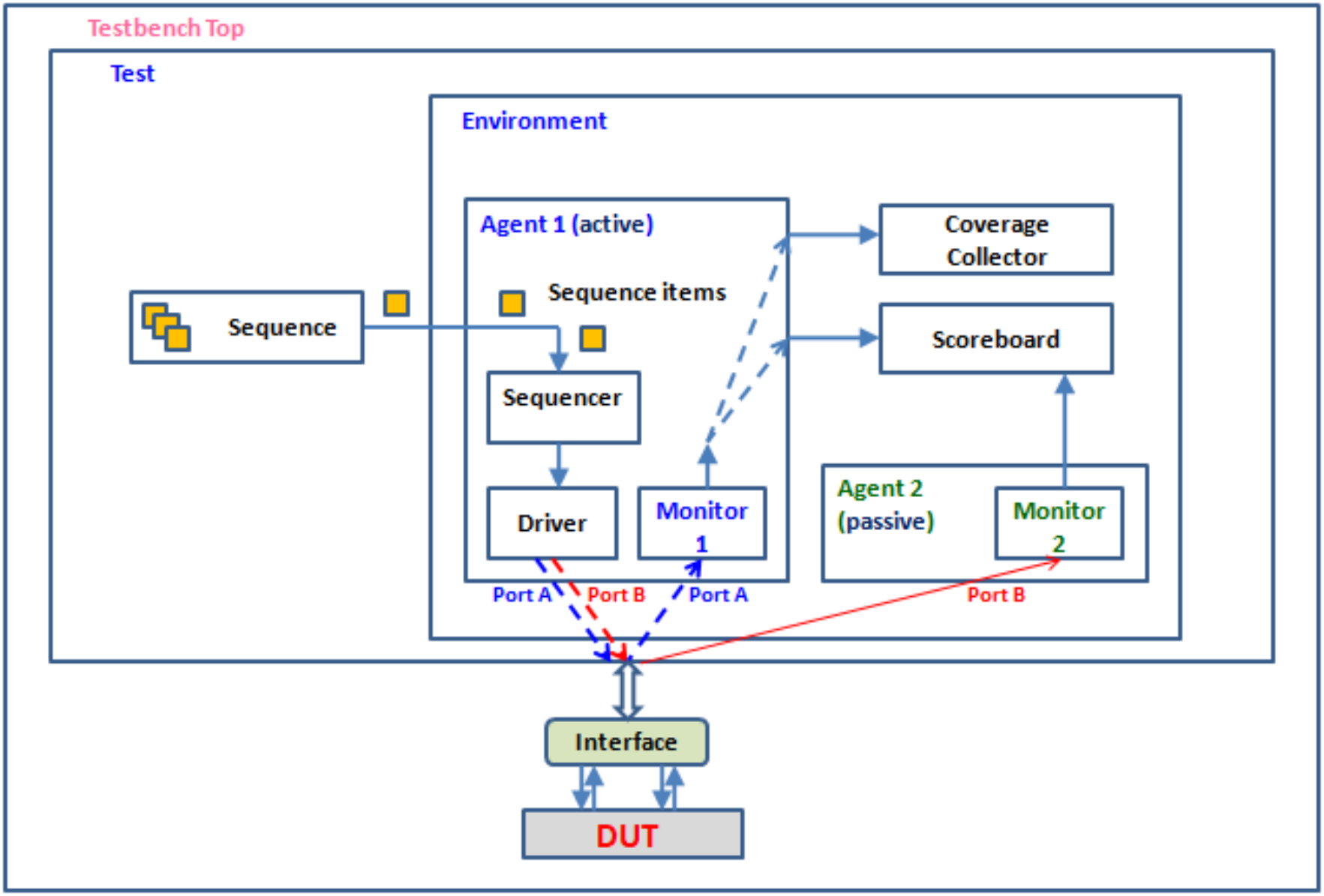

Renable
Wenable
WriteData
ReadData
WriteAddress
ReadAddress

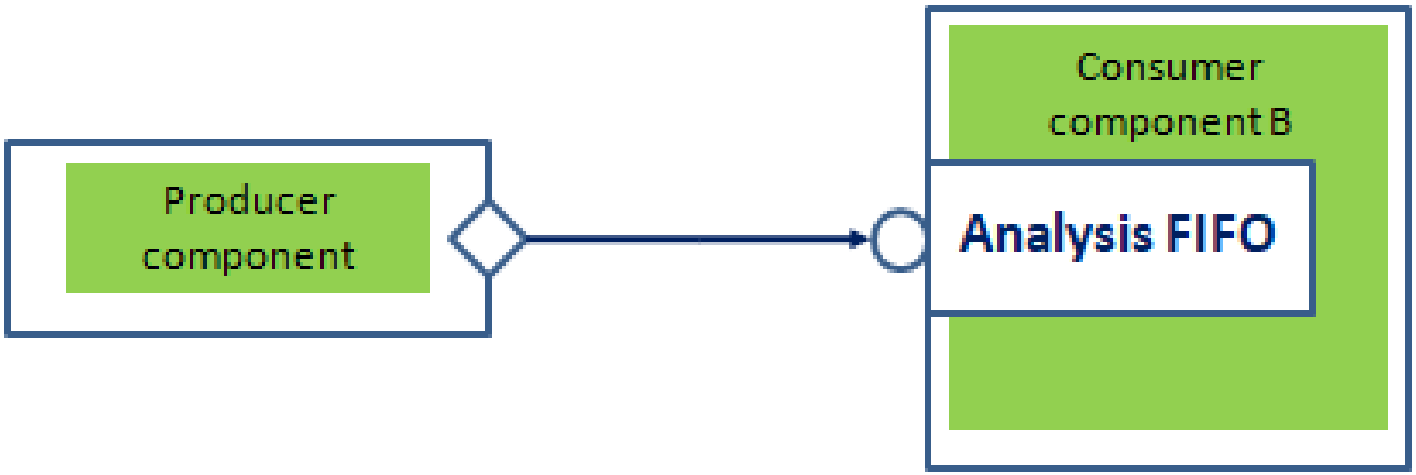
Typical Applications

It can be used in a variety of applications,

- such as in **communication systems**,
  - where data needs to be transferred
    - between multiple **devices** or **circuits** simultaneously.

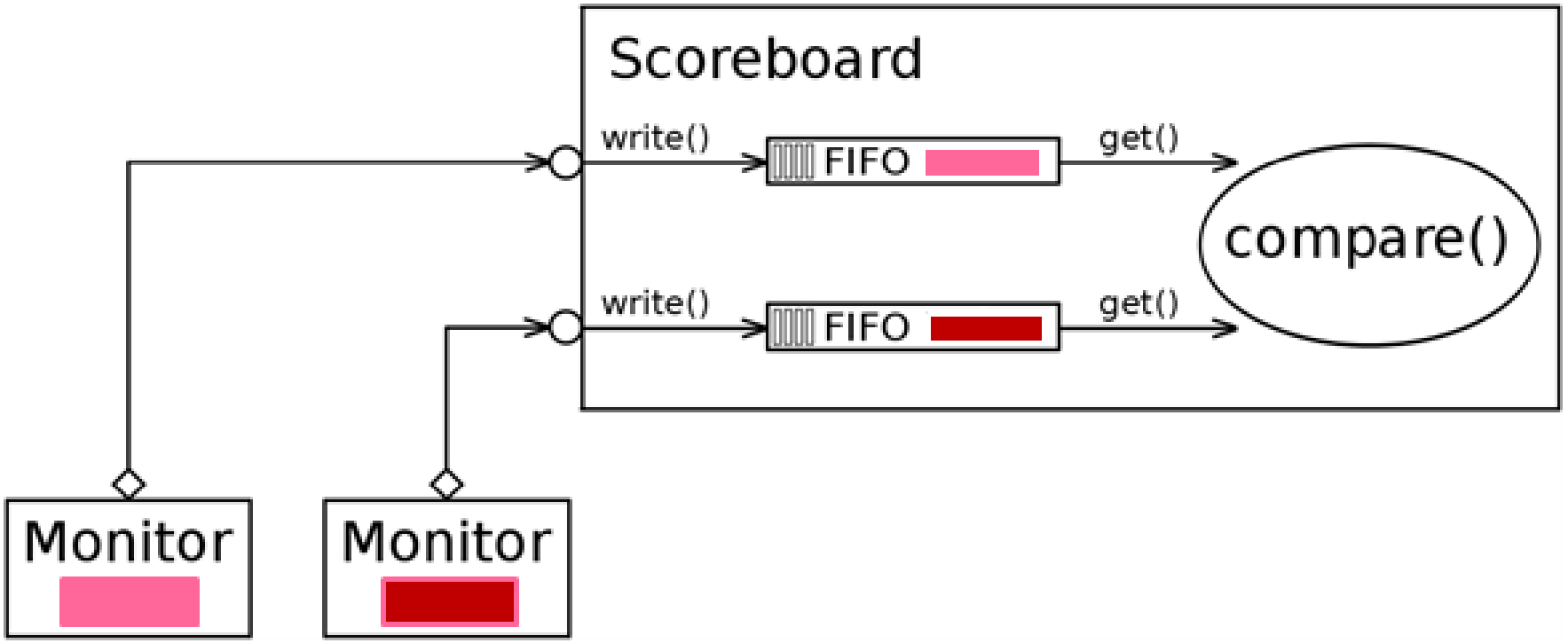
- It can also be used in **digital signal processing** (**DSP**) applications,
  - where data needs to be processed by multiple **circuits** in parallel.

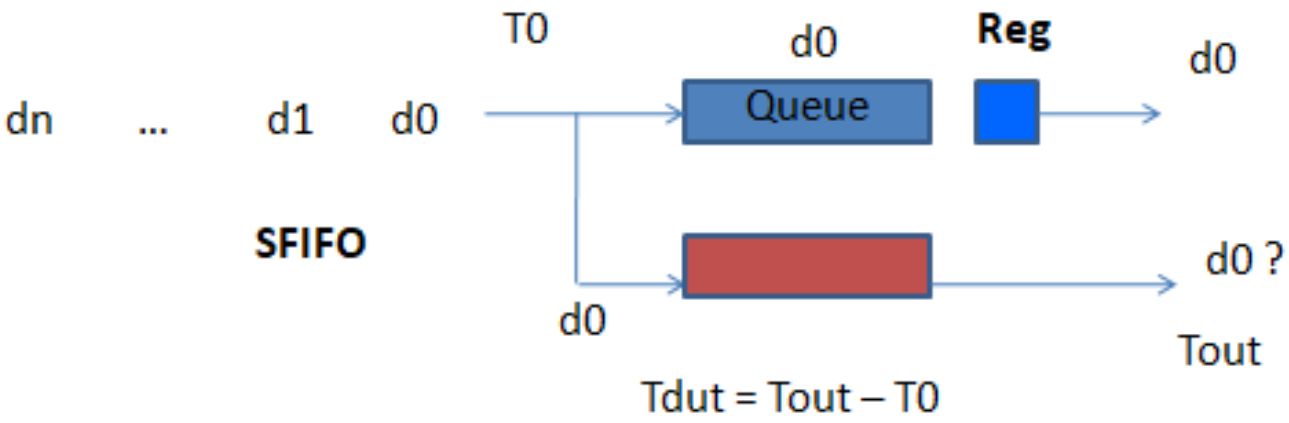

Testbench Top
Test
Environment
Agent 1 (active)
Sequence items
Sequence
Sequencer
Driver
Monitor 1
Port A
Port B
Port A
Coverage Collector
Scoreboard
Agent 2 (passive)
Monitor 2
Port B
Interface
DUT

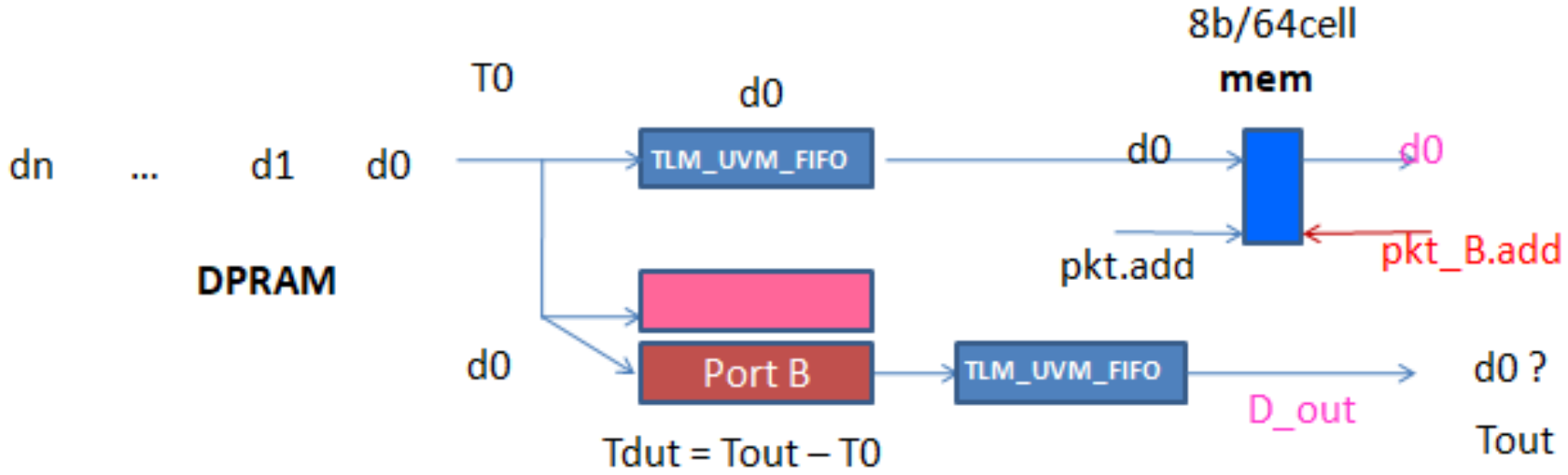


uvm_analysis_port #(seq_item)

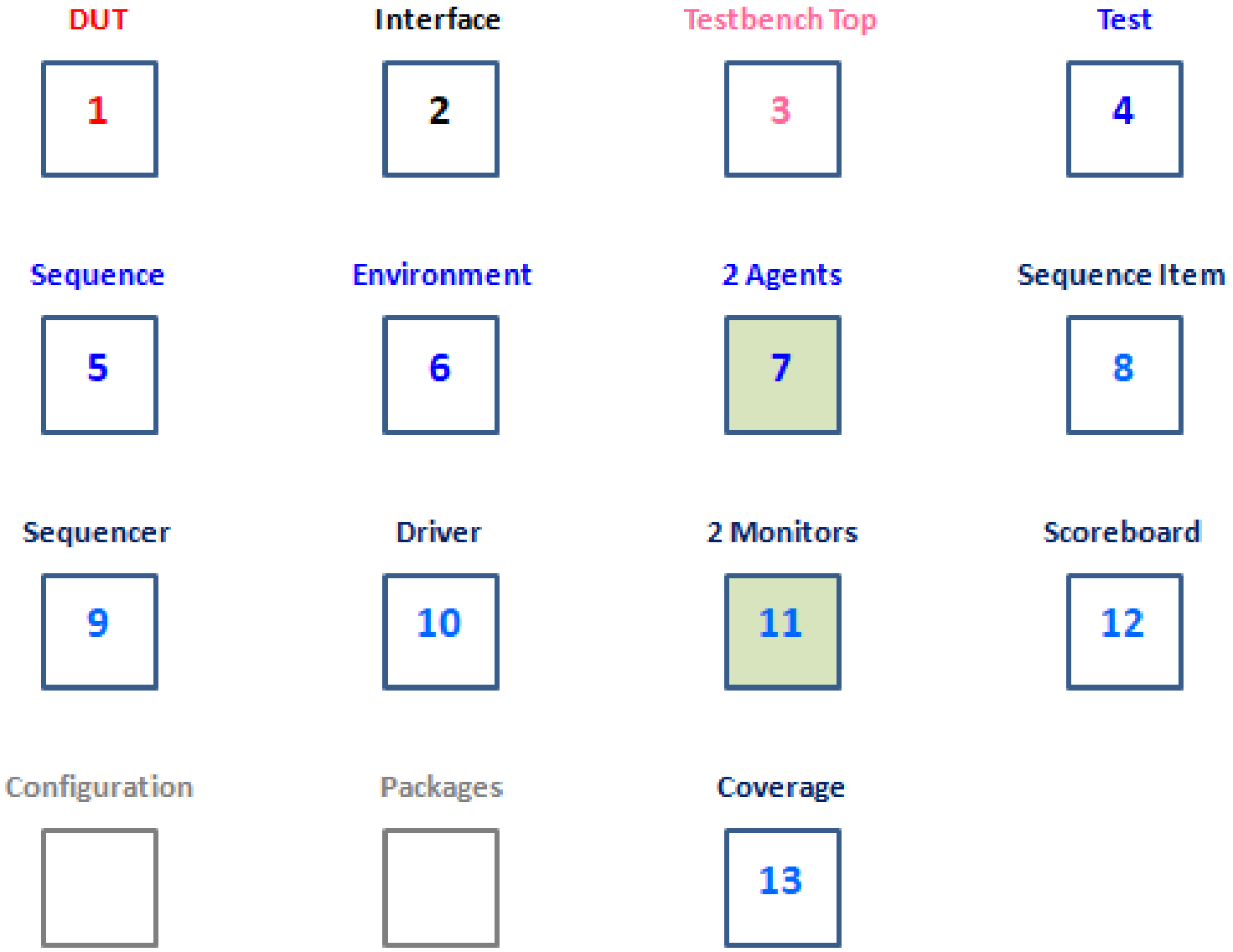
uvm_analysis_imp #(seq_item)


Scoreboard
write()
FIFO
get()
compare()
write()
FIFO
get()
Monitor
Monitor


T0
d0
Reg
d0
dn
...
d1
d0
Queue
SFIFO
d0 ?
d0
Tout
Tdut = Tout – T0


8b/64cell
mem
T0
d0
dn
...
d1
d0
TLM_UVM_FIFO
d0
d0
pkt.add
pkt_B.add
DPRAM
d0
Port B
TLM_UVM_FIFO
d0 ?
D_out
Tout
Tdut = Tout – T0

DUT and UVM Testbench – Basic



Implementation
Onsite Practice

Hint

- Review:
  - PKG
  - SEQ/COV/SB
  - CONN
  - CFG
  - ARCH

DUT

```
module tdp_ram (
                clk,
                din_a,  // data in at port A
                din_b,  // data in at port B
                dout_a, // data out at port A
                dout_b, // data out at port B
                addr_a, // address in at port A
                addr_b, // address in at port B
                we_a,   // write enable at port A
                we_b,   // write enable at port B
                sclr    // synchronous clear/reset port
              );

input clk;
input [7:0] din_a, din_b;
output [7:0] dout_a, dout_b;
reg [7:0] dout_a, dout_b;
input [5:0] addr_a, addr_b;
input we_a, we_b;
input sclr;
```

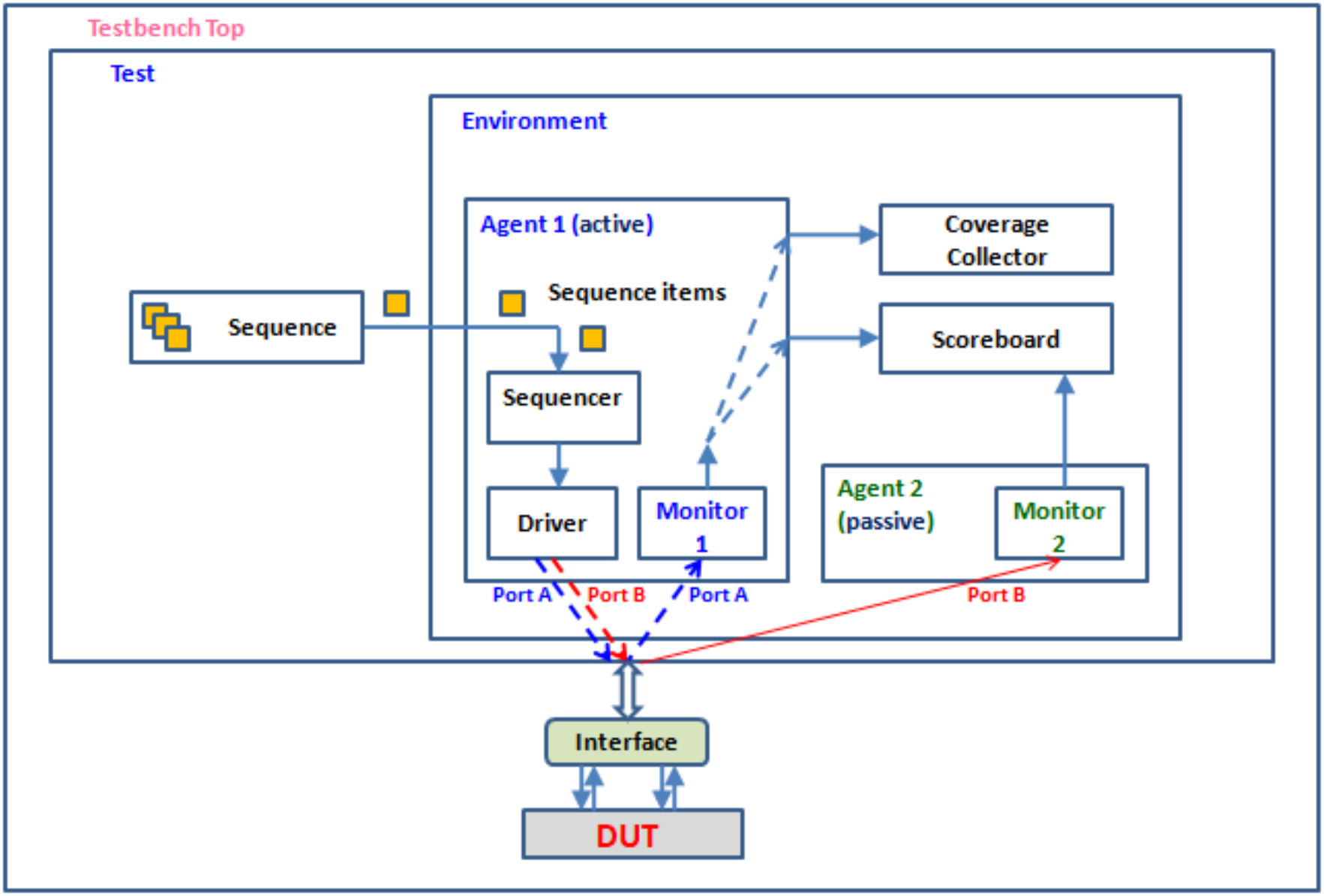
```
interface tdpram_interface (input bit clk);

logic [7:0] din_a;
logic [7:0] dout_a;
logic [5:0] addr_a;
logic we_a;
logic [7:0] din_b;
logic [7:0] dout_b;
logic [5:0] addr_b;
logic we_b;
logic sclr;

modport tb (
            output clk, sclr, we_a, we_b, din_a, din_b, addr_a, addr_b,
            input dout_a, dout_b
            );

modport dut ( // RTL
            input clk, sclr, we_a, we_b, din_a, din_b, addr_a, addr_b,
            output dout_a, dout_b
            );

endinterface
```


Testbench Top
Test
Environment
Agent 1 (active)
Sequence items
Sequence
Sequencer
Driver
Monitor 1
Coverage Collector
Scoreboard
Agent 2 (passive)
Monitor 2
Port A
Port B
Port A
Port B
Interface
DUT

ARCH

```
class tdpram_drv extends uvm_driver #(tdpram_seq_item);
 `uvm_component_utils(tdpram_drv)
 // Further code here

function new(string name="tdpram_drv",uvm_component parent);
   super.new(name,parent);
endfunction

 // Further code here

endclass
```

```
class tdpram_mon1 extends uvm_monitor;
`uvm_component_utils(tdpram_mon1)

// Futher code here

function new(string name="tdpram_mon1",uvm_component parent);
  super.new(name,parent);
  // Futher code here
endfunction

// Futher code here

endclass
```

```
class tdpram_mon2 extends uvm_monitor;
 `uvm_component_utils(tdpram_mon2)

 // Further code here

function new(string name="tdpram_mon2",uvm_component parent);
   super.new(name,parent);
   // Further code here
endfunction

 // Further code here

endclass
```

```
class tdpram_seqr extends uvm_sequencer#(tdpram_seq_item);

`uvm_component_utils(tdpram_seqr)

function new(string name="tdpram_seqr",uvm_component parent);
  super.new(name,parent);
endfunction

endclass
```

```
class tdpram_agent1 extends uvm_agent;
`uvm_component_utils(tdpram_agent1)

tdpram_seqr seqr;
tdpram_drv drv;
tdpram_mon1 mon1;

function new(string name="tdpram_agent1",uvm_component parent);
  super.new(name,parent);
endfunction

virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  seqr=tdpram_seqr::type_id::create("seqr",this);
  drv=tdpram_drv::type_id::create("drv",this);
  mon1=tdpram_mon1::type_id::create("mon1",this);
endfunction

// Further code here

endclass
```

```
class tdpram_agent2 extends uvm_agent;
`uvm_component_utils(tdpram_agent2)

tdpram_mon2 mon2;

function new(string name="tdpram_agent2",uvm_component parent);
  super.new(name,parent);
endfunction

virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  mon2=tdpram_mon2::type_id::create("mon2",this);
endfunction

endclass
```

```
class tdpram_sb extends uvm_scoreboard;
`uvm_component_utils(tdpram_sb);
// Further code here

function new(string name="tdpram_sb",uvm_component parent);
  super.new(name,parent);
  // Further code here
endfunction

virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  // Further code here
endfunction

// Further code here

endclass
```

```
virtual class uvm_subscriber #(type T=int) extends uvm_component;

  typedef uvm_subscriber #(T) this_type;
  uvm_analysis_imp #(T, this_type) analysis_export;

  function new (string name, uvm_component parent);
    super.new(name, parent);
    analysis_export = new("analysis_imp", this);
  endfunction

  pure virtual function void write(T t);
endclass
```

```
class tdpram_cov extends uvm_subscriber #(// data transaction);
`uvm_component_utils(tdpram_cov)
  // data transaction

  covergroup cov_inst;
  // Coverage insertion
  endgroup


function new(string name="",uvm_component parent);
  super.new(name,parent);
  cov_inst = new();
endfunction

function void build_phase(uvm_phase phase);
  super.build_phase(phase);
endfunction

// Further code here

endclass
```

```
class tdpram_env extends uvm_env;
`uvm_component_utils(tdpram_env)

tdpram_agent1 agent1;
tdpram_agent2 agent2;
tdpram_sb sb;
tdpram_cov cov;

function new(string name="tdpram_env",uvm_component parent);
  super.new(name,parent);
endfunction

virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  agent1 = tdpram_agent1::type_id::create("agent1",this);
  agent2 = tdpram_agent2::type_id::create("agent2",this);
  sb = tdpram_sb::type_id::create("sb",this);
  cov = tdpram_cov::type_id::create("cov", this);
endfunction

// Further code here

endclass
```

```
class tdpram_seq extends uvm_sequence#(// data transaction);
`uvm_object_utils(tdpram_seq)
// data transaction

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

// Further code here

endclass
```

```
class tdpram_test extends uvm_test;
`uvm_component_utils(tdpram_test)
tdpram_seq seq;
tdpram_env env;

function new(string name="tdpram_test",uvm_component parent);
  super.new(name,parent);

endfunction

virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  seq = tdpram_seq::type_id::create("seq", this);
  env = tdpram_env::type_id::create("env", this);
endfunction

// Further code here

endclass
```

CFG

```
class tdpram_drv extends uvm_driver #(//data transaction);
`uvm_component_utils(tdpram_drv)
virtual tdpram_interface inf; //

function new(string name="tdpram_drv",uvm_component parent);
  super.new(name,parent);
endfunction

virtual function void build_phase(uvm_phase phase); //
  super.build_phase(phase); //
  uvm_config_db #(virtual tdpram_interface)::get(this, "", "inf", inf); //
endfunction //

// Further code here

endclass
```

```
class tdpram_mon1 extends uvm_monitor;
`uvm_component_utils(tdpram_mon1)
virtual tdpram_interface inf; //
// data transaction

function new(string name="tdpram_mon1",uvm_component parent);
  super.new(name,parent);
  // Further code here
endfunction

virtual function void build_phase(uvm_phase phase); //
  super.build_phase(phase); //
  uvm_config_db #(virtual tdpram_interface) ::get(this,"","inf",inf); //
endfunction //

// Further code here

endclass
```

```
class tdpram_mon2 extends uvm_monitor;
`uvm_component_utils(tdpram_mon2)
virtual tdpram_interface inf;
// data transaction

function new(string name="tdpram_mon2",uvm_component parent);
  super.new(name,parent);
  // Further code here
endfunction

virtual function void build_phase(uvm_phase phase); //
  super.build_phase(phase); //
  uvm_config_db #(virtual tdpram_interface)::get(this,"","inf",inf); //
endfunction //

// Further code here

endclass
```

CONN

```
class tdpram_seq_item extends uvm_sequence_item;
`uvm_object_utils(tdpram_seq_item)

rand bit[7:0] data_in;
rand bit[5:0] add;
bit we;
bit sclr;
bit[7:0] data_out;

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

endclass
```

Driver

```
task run_phase(uvm_phase phase);
  tdpram_seq_item pkt;
  pkt = tdpram_seq_item::type_id::create("pkt");
  forever
    begin
      seq_item_port.get_next_item(pkt);
      pkt.sclr = 1;
      @(negedge inf.clk);
      inf.sclr = pkt.sclr;
      if(pkt.we == 0) begin
        inf.we_a = pkt.we;
        inf.din_a = pkt.data_in;
        inf.addr_a = pkt.add;
        `uvm_info("DRV TRANSACTIONS", $sformatf("inf.din_a=%d,inf.addr_a=%d,
        inf.we_a=%b",inf.din_a,inf.addr_a,inf.we_a) ,UVM_NONE);
      end
      else begin
        inf.we_b = pkt.we;
        inf.addr_b = pkt.add;
        `uvm_info("DRV TRANSACTIONS", $sformatf("inf.addr_b=%d, inf.we_b=%b",
        inf.addr_b,inf.we_b) ,UVM_NONE);
      end
      @(negedge inf.clk);
      seq_item_port.item_done();
      `uvm_info("DRV","DRV TRANSACTION TO DUT",UVM_NONE);
      #5;
    end
endtask
```

mon1

```
class tdpram_mon1 extends uvm_monitor;
`uvm_component_utils(tdpram_mon1)
virtual tdpram_interface inf;
tdpram_seq_item pkt; //

uvm_analysis_port #(tdpram_seq_item) item_collected_port; //
uvm_analysis_port #(tdpram_seq_item) cov_ap; //

function new(string name="tdpram_mon1",uvm_component parent);
  super.new(name,parent);
  item_collected_port = new("item_collected_port",this); //
  cov_ap = new("analysis_port",this); //
endfunction

virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  uvm_config_db #(virtual tdpram_interface) ::get(this,"","inf",inf);
endfunction
```

```
virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  uvm_config_db #(virtual tdpram_interface) ::get(this,"","inf",inf);
endfunction

task run_phase(uvm_phase phase);
  pkt = tdpram_seq_item::type_id::create("pkt");
  forever
    begin
      #5;
      @(posedge inf.clk)
      if((inf.sclr == 1) && (inf.we_a == 0)) begin
        pkt.sclr = inf.sclr;
        pkt.we = inf.we_a;
        pkt.data_in = inf.din_a;
        pkt.add = inf.addr_a;
        `uvm_info("MON1","MON1 TRANSACTIONS" ,UVM_NONE);
      end
      `uvm_info("MON1","MON1 TRANSACTIONS",UVM_NONE);
      @(posedge inf.clk)
      item_collected_port.write(pkt);
      cov_ap.write (pkt);
    end
endtask

endclass
```

mon2

```
class tdpram_mon2 extends uvm_monitor;
`uvm_component_utils(tdpram_mon2)
tdpram_seq_item pkt_B;
uvm_analysis_port #(tdpram_seq_item)item_collected_port_B;

function new(string name="tdpram_mon2",uvm_component parent);
  super.new(name,parent);
  item_collected_port_B=new("item_collected_port_B",this);
endfunction

virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  uvm_config_db #(virtual tdpram_interface)::get(this,"","inf",inf);
endfunction
```

```
task run_phase(uvm_phase phase);
  pkt_B = tdpram_seq_item::type_id::create("pkt_B");
  forever
    begin
      #2;
      @(posedge inf.clk) begin
        if (inf.we_b == 1) begin
          pkt_B.data_out = inf.dout_b;
          pkt_B.add=inf.addr_b;
          `uvm_info("MON2","MON2 TRANSACTIONS",UVM_NONE)
        end
      end
      `uvm_info("MON2","MON2 TRANSACTIONS",UVM_NONE)
      @(posedge inf.clk)
      item_collected_port_B.write(pkt_B);
    end
endtask

endclass
```

SB

```
class tdpram_sb extends uvm_scoreboard;
`uvm_component_utils(tdpram_sb);

tdpram_seq_item pkt;
tdpram_seq_item pkt_B;

bit [7:0] mem [0:63];

uvm_tlm_analysis_fifo #(tdpram_seq_item)ip_fifo; //
uvm_tlm_analysis_fifo #(tdpram_seq_item)op_fifo; //

function new(string name="tdpram_sb",uvm_component parent);
  super.new(name,parent);
  ip_fifo = new("ip_fifo",this);
  op_fifo = new("op_fifo",this);
endfunction

virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  pkt   =tdpram_seq_item::type_id::create("pkt", this);
  pkt_B =tdpram_seq_item::type_id::create("pkt_B",this);
endfunction
```

```
task run_phase(uvm_phase phase);
  forever
    begin
      fork
        ip_fifo.get(pkt);
        `uvm_info("SB","TRANSACTIONS FROM MON1",UVM_NONE);
        $display("sb data_in=%d,add=%d",pkt.data_in,pkt.add);
        op_fifo.get(pkt_B);
        `uvm_info("SB","TRANSACTIONS FROM MON2",UVM_NONE);
        $display("sb data_out=%d,add=%d",pkt_B.data_out,pkt_B.add);
      join
      if(pkt.we==0)
        begin
          mem[pkt.add]=pkt.data_in;
        end

      if(pkt_B.data_out==mem[pkt_B.add])
        begin
          `uvm_info("SB MATCHED",$sformatf("DATA pkt.din_a=%d,pkt_B.dout_b=%d",mem[pkt.add],pkt_B.data_out),UVM_NONE);
        end
      else
        begin
          `uvm_info("SB NOT MATCHED",$sformatf("DATA pkt.din_a=%d,pkt_B.dout_b=%d",mem[pkt.add],pkt_B.data_out),UVM_NONE);
        end
    end
endtask
```

Cov

```
class tdpram_cov extends uvm_subscriber #(tdpram_seq_item)
`uvm_component_utils(tdpram_cov)
tdpram_seq_item trans;

  covergroup cov_inst;
  ADDRESS:coverpoint trans.add {option.auto_bin_max = 6;}
  DATA:coverpoint trans.data_in {option.auto_bin_max = 8;}
  endgroup

function new(string name="",uvm_component parent);
  super.new(name,parent);
  cov_inst = new();
endfunction

function void build_phase(uvm_phase phase);
  super.build_phase(phase);
endfunction

virtual function void write(tdpram_seq_item t);
  $cast(trans, t);
  cov_inst.sample();
endfunction

endclass
```

env

```
class tdpram_env extends uvm_env;
`uvm_component_utils(tdpram_env)

tdpram_agent1 agent1;
tdpram_agent2 agent2;
tdpram_sb sb;
tdpram_cov cov;

function new(string name="tdpram_env",uvm_component parent);
  super.new(name,parent);
endfunction

virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  agent1 = tdpram_agent1::type_id::create("agent1",this);
  agent2 = tdpram_agent2::type_id::create("agent2",this);
  sb = tdpram_sb::type_id::create("sb",this);
  cov = tdpram_cov::type_id::create("cov", this);
endfunction

virtual function void connect_phase(uvm_phase phase);
  super.connect_phase(phase);
  agent1.mon1.item_collected_port.connect(sb.ip_fifo.analysis_export);
  agent1.mon1.cov_ap.connect(cov.analysis_export);
  agent2.mon2.item_collected_port_B.connect(sb.op_fifo.analysis_export);
endfunction
```

seq

```
class tdpram_seq extends uvm_sequence#(tdpram_seq_item);
 `uvm_object_utils(tdpram_seq)
 tdpram_seq_item pkt;

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

task body();
   pkt = tdpram_seq_item::type_id::create("pkt");
   repeat(10)  begin
     //
     start_item(pkt);
     assert(pkt.randomize());
     pkt.we = 0;
     pkt.print();
     finish_item(pkt);
     //
     start_item(pkt);
     pkt.we=1;
     pkt.print();
     finish_item(pkt);
     `uvm_info("SEQ","SEQUENCE TRANSACTIONS",UVM_NONE);
   end
endtask

endclass
```

```
package tdpram_tb_pkg;

  `include "uvm_macros.svh"
  import uvm_pkg::*;

  // include files:
  `include "tdpram_seq_item.svh"
  `include "tdpram_seq.svh"
  `include "tdpram_seqr.svh"
  `include "tdpram_drv.svh"
  `include "tdpram_cov.svh"
  `include "tdpram_mon1.svh"
  `include "tdpram_mon2.svh"
  `include "tdpram_agent1.svh"
  `include "tdpram_agent2.svh"
  `include "tdpram_sb.svh"
  `include "tdpram_env.svh"
  `include "tdpram_test.svh"

endpackage
```

test

```
class tdpram_test extends uvm_test;
`uvm_component_utils(tdpram_test)
tdpram_seq seq;
tdpram_env env;


function new(string name="tdpram_test",uvm_component parent);
  super.new(name,parent);

endfunction

virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  seq = tdpram_seq::type_id::create("seq", this);
  env = tdpram_env::type_id::create("env", this);
endfunction

task run_phase(uvm_phase phase);

  phase.raise_objection(this);
  seq.start(env.agent1.seqr);
  #50;
  phase.drop_objection(this);

endtask

endclass
```

tb_top

```
import uvm_pkg::*;
`include "uvm_macros.svh"

`include "tdpram_interface.sv"
`include "tdp_ram.v"
`include "tdpram_tb_pkg.sv"

module tb_top();

import tdpram_tb_pkg::*;

bit clk;

initial begin
    clk = 1'b1;
end

always #5 clk = ~clk;

tdpram_interface inf (clk);
```

```
tdp_ram dut (.clk(inf.clk),
             .din_a(inf.din_a),
             .dout_a(inf.dout_a),
             .addr_a(inf.addr_a),
             .we_a(inf.we_a),
             .din_b(inf.din_b),
             .dout_b(inf.dout_b),
             .addr_b(inf.addr_b),
             .we_b(inf.we_b),
             .sclr(inf.sclr)
            );

initial begin
uvm_config_db#(virtual tdpram_interface)::set(null,"*","inf",inf);
run_test();
end

endmodule
```

```
vlib work
vmap work work

# Compilation: ----------------------------------------------------

vlog tdp_ram.v
#vlog sfifo_interface.sv
#vlog sfifo_agent_pkg.sv
#vlog sfifo_environment_pkg.sv
#vlog sfifo_sequence_pkg.sv
#vlog sfifo_test_pkg.sv
vlog tb_top.sv

# Simulation: ----------------------------------------------------
#vsim -coverage tb_top +UVM_TESTNAME=tdpram_test
vopt work.tb_top -o top_optimized +acc +cover=sbfect
vsim -c top_optimized -coverage +UVM_TESTNAME=tdpram_test

# Coverage report: -----------------------------------------------
# Coverage save ucdb file:
coverage save -onexit -assert -directive -cvg -codeAll tdpram_test.ucdb

# Coverage reports with html and text files:
vcover report -html tdpram_test.ucdb -htmldir covhtmlreport
vcover report -file cov_report.txt tdpram_test.ucdb

add wave -r sim:/tb_top/*
run -all
```

Thank You