Appendix

Contact the authors for full source.

```
//=============================================
// File t.sv
//=============================================
import uvm_pkg::*;
`include "uvm_macros.svh"

import vip_pkg::*;
import test_pkg::*;


module top();
  m m_A();
  m m_B();

  initial begin
    int addr, rdata, wdata;

    m_A.sequencer_name = "uvm_test_top.e.a.sqr";
    m_A.driver_name    = "uvm_test_top.e.a.d";

    run_ph.wait_for_state( UVM_PHASE_STARTED,
UVM_GTE);

    run_ph.raise_objection(uvm_top);

    for(int i = 0; i < 10; i++) begin
      addr = i;
      wdata = addr*2;

      m_A.write(addr, wdata);
      m_A.read( addr, rdata);

      if (rdata != wdata) begin
        $display(
    "Error: Mismatch Addr %0d. rdata (%0d) !=
wdata (%0d)",
          addr, rdata, wdata);
      end
    end
    $display("Side A done");
    run_ph.drop_objection(uvm_top);
  end

  initial begin
    bit unsigned [31:0] crc;
    int addr, rdata, wdata;

    m_B.sequencer_name = "uvm_test_top.e.b.sqr";
    m_B.driver_name    = "uvm_test_top.e.b.d";

    run_ph.wait_for_state( UVM_PHASE_STARTED,
UVM_GTE);

    run_ph.raise_objection(uvm_top);

      m_B.walking_ones();
      m_B.dump();
      $display("CRC = %0x", crc);

      m_B.walking_zeroes();
      m_B.dump();
      $display("CRC = %0x", crc);

      begin : task_usage
        int read_count, write_count;
        m_B.rw_dist(1000, 20, 80,
          read_count, write_count);
       $display("rw_dist(1000, 20, 80). %0d
reads, %0d writes",
          read_count, write_count);
        m_B.dump();
      end   // task_usage

      begin : in_line_usage
        rw_dist_seq s;

        $display("Distribution of Reads and
Writes");
        if (m_B.sqr == null)

m_B.connect_to_sequencer(m_B.sequencer_name,
m_B.sqr);
        s = new("sequence");
        s.connection_name = $sformatf("%m");

        s.val.constraint_mode(0);
        s.dist_reads  = 20;
        s.dist_writes = 80;

        if (!s.randomize() with {
          number_of_transactions == 1000; } )
           `uvm_fatal("rw_dist", "Randomization
failed")

        s.start(m_B.sqr);
       $display("rw_dist(1000, 20, 80). %0d
reads, %0d writes",
          s.read_count, s.write_count);
        m_B.dump();
      end   // in_line_usage


    begin : rw_sequence_task_block
      bit unsigned [31:0] crc;
      bit unsigned [31:0] sum;

      for (int i = 0; i < 256-1; i++) begin
        m_B.rw_in_a_range(i, 256-i, i, crc);
        sum += crc;
      end
      $display("Sum=%0x", sum);
    end

    begin : rw_sequence_block
      rw_sequence_in_a_range s;
      bit unsigned [31:0] sum;

      s = new("sequence");
      for (int i = 0; i < 256-1; i++) begin
        s.low_addr = i;
        s.high_addr = 256-i;
        s.start(m_B.sqr);
        sum += s.crc;
      end
      $display("Sum=%0x", sum);
    end

    begin : rw_sequence_block_with_randomize
      rw_sequence_in_a_range s;
      bit unsigned [31:0] sum;

      s = new("sequence");
      for (int i = 0; i < 256; i++) begin
        if (!s.randomize())
          `uvm_fatal("rw_dist",
            {"Randomization failed (",
             s.convert2string(),
             ")"})
        s.start(m_B.sqr);
        sum += s.crc;
      end
      $display("Sum=%0x", sum);
    end

    $display("Side B done");
    run_ph.drop_objection(uvm_top);
  end

  initial
    run_test("test");

endmodule


//=============================================
// File module_connector.sv
//=============================================
import uvm_pkg::*;
`include "uvm_macros.svh"

import vip_pkg::*;
import test_pkg::*;

// Macro for Boilerplate
//  1) Is the sequencer already set?
//     If not, then look it up.
//  2) Construct the 'seq'.
//  3) Set an internal field to know where this
//     transaction came from.
`define SEQ_INIT(seq) \
  if (sqr == null) \
    connect_to_sequencer(sequencer_name, sqr); \
  seq = new("sequence"); \
  seq.connection_name = $sformatf("%m"); \


module m();

  string sequencer_name;

  uvm_sequencer_base sqr;


  task read(int addr, output int data);
    read_seq s;

    `SEQ_INIT(s)

    s.addr = addr;
    s.start(sqr);
    data = s.data;
  endtask

  task write(int addr, int data);
    write_seq s;

    `SEQ_INIT(s)

    s.addr = addr;
    s.data = data;
    s.start(sqr);
  endtask

  task rw_in_a_range(int low_addr,
         int high_addr,
         int data,
   output bit [31:0] crc);

    rw_sequence_in_a_range s;
    $display("INFO: Reads and Writes in a
Range");
    `SEQ_INIT(s)
    s.low_addr = low_addr;
    s.high_addr = high_addr;
    s.data = data;
    s.start(sqr);
    crc = s.crc;
  endtask


  task rw_dist(
          int number_of_transactions,
          int dist_reads,
          int dist_writes,
    output int read_count,
    output int write_count
  );
    rw_dist_seq s;

    $display("INFO: Distribution of Reads and
Writes");
    `SEQ_INIT(s)

    s.val.constraint_mode(0);
    s.dist_reads  = dist_reads;
    s.dist_writes = dist_writes;

    if (!s.randomize() with {
      number_of_transactions ==
local::number_of_transactions;
      } )
      `uvm_fatal("rw_dist", "Randomization
failed")

    s.start(sqr);
    read_count  = s.read_count;
    write_count = s.write_count;
  endtask


  task walking_zeroes(int number_of_writes =
256);
    walking_zeroes_seq s;

    $display("INFO: Walking Zeroes");
    `SEQ_INIT(s)

    s.number_of_writes = number_of_writes;
    s.start(sqr);
  endtask


  task walking_ones(int number_of_writes = 256);
    walking_ones_seq s;

    $display("INFO: Walking Ones");
    `SEQ_INIT(s)

    s.number_of_writes = number_of_writes;
    s.start(sqr);
  endtask


  // Utility/Debug Code
  string driver_name;
  driver d;

  task dump();
    if (d == null)
      $cast(d, uvm_top.find(driver_name));
    d.dump();
  endtask


  task automatic connect_to_sequencer(
      string sequencer_name = "sequencer",
      ref uvm_sequencer_base sqr);

    if (sqr == null) begin
      uvm_phase run_phase;

      if (0) begin
        // Lookup by name.
        uvm_domain common_domain;
        common_domain =
uvm_domain::get_common_domain() ;
        run_phase  =
common_domain.find_by_name("run");
        // Or just 'get' it.
        run_phase = uvm_run_phase::get();
      end

      // Or just use a global.
      run_phase  = run_ph;

      run_phase.wait_for_state(
UVM_PHASE_STARTED, UVM_GTE);

      $cast(sqr, uvm_top.find(sequencer_name));

      `uvm_info($sformatf("%m"),
        $sformatf("Module %m connected to
sequencer '%s'",
        sqr.get_full_name()), UVM_MEDIUM)
    end
  endtask
endmodule



//=============================================
// File vip_pkg.sv
//=============================================

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

  `include "transaction.svh"
  `include "seq_lib.svh"
  `include "driver.svh"
  `include "agent.svh"
  `include "env.svh"
endpackage


//=============================================
// File transaction.svh
//=============================================

typedef enum bit[2:0] {IDLE, READ, WRITE } rw_t;


class my_transaction extends uvm_sequence_item;
  `uvm_object_utils(my_transaction)

  string connection_name = "unconnected";

  rand rw_t       rw;
  rand bit [31:0] addr;
  rand bit [31:0] data;

  constraint addr_value { addr >= 0; addr <
1000; }
  //constraint data_value { data >= 0; data <
1000; }

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

  function string convert2string();
    return $sformatf("%s(%0x, %0x) [--%s--]",
      rw.name(),
      addr,
      data,
      connection_name);
  endfunction
endclass

class read_transaction extends my_transaction;
  `uvm_object_utils(read_transaction)

  constraint rw_value { rw == READ; }

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

class write_transaction extends my_transaction;
  `uvm_object_utils(write_transaction)

  constraint rw_value { rw == WRITE; }

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


//===============================================
// File seq_lib.svh
//===============================================
virtual class seq extends
uvm_sequence#(my_transaction);
  `uvm_object_utils(seq)

  string connection_name = "unconnected";

  rand bit [31:0] addr;
  rand bit [31:0] data;

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


class read_seq extends seq;
  `uvm_object_utils(read_seq)

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

  task body();
    read_transaction read_t;

    read_t = new("read_t");
    read_t.connection_name = connection_name;
    if (!read_t.randomize()
      with {addr == local::addr;}
    )
      `uvm_fatal(get_type_name(), "Randomization
failed")
    `uvm_info(get_type_name(),
$sformatf("Sending %s",
      read_t.get_type_name()), UVM_HIGH)
    start_item(read_t);
    finish_item(read_t);
    data = read_t.data;
  endtask
endclass


class write_seq extends seq;
  `uvm_object_utils(write_seq)

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

  task body();
    write_transaction write_t;

    write_t = new("write_t");
    write_t.connection_name = connection_name;
    if (!write_t.randomize()
      with {data == local::data;
            addr == local::addr;}
    )
      `uvm_fatal(get_type_name(), "Randomization
failed")
    `uvm_info(get_type_name(),
$sformatf("Sending %s",
      write_t.get_type_name()), UVM_HIGH)
    start_item(write_t);
    finish_item(write_t);
  endtask
endclass

class rw_sequence_in_a_range extends seq;
  `uvm_object_utils(rw_sequence_in_a_range)

  rand bit [31:0] low_addr;
  rand bit [31:0] high_addr;

  rand bit [31:0] data;

  bit unsigned [31:0] crc;

  constraint val {
    low_addr < high_addr;
    low_addr >= 0;
    high_addr < 256;
  }

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

  task body();
    read_transaction  read_t;
    write_transaction write_t;

    read_t  = new("read_t");
    write_t = new("write_t");
    read_t.connection_name  = connection_name;
    write_t.connection_name = connection_name;

    crc = 0;
    for (int addr = low_addr;
             addr < high_addr;
             addr++ ) begin

      if (!write_t.randomize()
        with {data == local::data;
              addr == local::addr;})
        `uvm_fatal(get_type_name(),
          {"Randomization failed",
           write_t.convert2string()})

      start_item(write_t);
      finish_item(write_t);

      if (!read_t.randomize()
        with {addr == local::addr;})
        `uvm_fatal(get_type_name(),
"Randomization failed")

      start_item(read_t);
      finish_item(read_t);

      if (read_t.data != write_t.data)
        `uvm_warning(get_type_name(),
          $sformatf("Mismatch: wrote %0x, read
%0x",
            write_t.data, read_t.data))
      crc = (crc<<1) ^ (crc) + read_t.data;
    end
  endtask
endclass


class walking_ones_seq extends seq;
  `uvm_object_utils(walking_ones_seq)

  rand int number_of_writes;

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

  task body();
    bit unsigned [31:0] my_data;
    write_transaction write_t;

    write_t = new("write_t");
    write_t.connection_name = connection_name;

    // Turn off data constraint
    write_t.data_value.constraint_mode(0);

    my_data = 1;
    for (int i = 0; i < number_of_writes; i++)
begin
      if (!write_t.randomize()
        with {data == my_data;
              addr == i;}
      )
        `uvm_fatal(get_type_name(),
"Randomization failed")
      `uvm_info(get_type_name(),
$sformatf("Sending %s",
        write_t.get_type_name()), UVM_HIGH)
      start_item(write_t);
      finish_item(write_t);
      my_data = my_data << 1;
      if (my_data == 0)
        my_data = 1;
    end
  endtask
endclass


class walking_zeroes_seq extends seq;
  `uvm_object_utils(walking_zeroes_seq)

  rand int number_of_writes;

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

  task body();
    int j;
    bit unsigned [31:0] my_data;
    write_transaction write_t;

    write_t = new("write_t");
    write_t.connection_name = connection_name;

    // Turn off data constraint
    write_t.data_value.constraint_mode(0);

    for (int i = 0; i < number_of_writes; i++)
begin
      j = i % 32;
      my_data = '1;
      my_data[j] = 0;
      if (!write_t.randomize()
        with {data == my_data;
              addr == i;}
      )
        `uvm_fatal(get_type_name(),
"Randomization failed")
      `uvm_info(get_type_name(),
$sformatf("Sending %s",
        write_t.get_type_name()), UVM_HIGH)
      start_item(write_t);
      finish_item(write_t);
    end
  endtask
endclass


class random_read_write;
  int dist_reads;
  int dist_writes;

  rand rw_t rw;

  constraint type_constraint {
    rw inside {READ, WRITE};
    rw dist { READ := dist_reads, WRITE :=
dist_writes };
  }
endclass

class rw_dist_seq extends seq;
  `uvm_object_utils(rw_dist_seq)

  rand int number_of_transactions;
       int dist_reads;
       int dist_writes;
       int read_count;
       int write_count;

  constraint val {
    number_of_transactions > 2;
    number_of_transactions < 1000;
  }

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

  task body();
    random_read_write random_rw;
    read_count = 0;
    write_count = 0;

    random_rw = new();
    random_rw.dist_reads = dist_reads;
    random_rw.dist_writes = dist_writes;

    for(int i = 0; i < number_of_transactions;
i++) begin
      if (!random_rw.randomize())
        `uvm_fatal(get_type_name(),
"Randomization failed")

      if (random_rw.rw == READ) begin
        read_transaction  read_t;
        read_t = new("read_t");
        read_t.connection_name =
connection_name;
        if (!read_t.randomize())
          `uvm_fatal(get_type_name(),
"Randomization failed")
        `uvm_info(get_type_name(),
$sformatf("Sending %s",
          read_t.get_type_name()), UVM_HIGH)
        start_item(read_t);
        finish_item(read_t);
        read_count++;
      end
      else if (random_rw.rw == WRITE) begin
        write_transaction write_t;
        write_t = new("write_t");
        write_t.connection_name =
connection_name;
        if (!write_t.randomize())
          `uvm_fatal(get_type_name(),
"Randomization failed")
        `uvm_info(get_type_name(),
$sformatf("Sending %s",
          write_t.get_type_name()), UVM_HIGH)
        start_item(write_t);
        finish_item(write_t);
        write_count++;
      end
    end

    $display(
     "reads=%0d [%f%%], writes=%0d [%f%%]",
      read_count,
      read_count  * 100 /
       (read_count+write_count),
      write_count,
      write_count * 100 /
       (read_count+write_count));
  endtask
endclass



//===============================================
// File test.sv
//===============================================

package test_pkg;

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

  import vip_pkg::*;

  class test extends uvm_test;
    `uvm_component_utils(test)

    env e;

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

    function void build_phase(uvm_phase phase);
      e = new("e", this);
    endfunction
  endclass

endpackage


//===============================================
// File env.svh
//=============================================

class env extends uvm_env;
  `uvm_component_utils(env)

  agent a;
  agent b;

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

  function void build_phase(uvm_phase phase);
    a = new("a", this);
    b = new("b", this);
  endfunction
endclass


//=============================================
// File agent.svh
//=============================================

class agent extends uvm_agent;
  `uvm_component_utils(agent)

  driver d;
  uvm_sequencer #(my_transaction) sqr;

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

  function void build_phase(uvm_phase phase);
    d   = new("d",   this);
    sqr = new("sqr", this);
  endfunction

  function void connect_phase(uvm_phase phase);

d.seq_item_port.connect(sqr.seq_item_export);
  endfunction
endclass


//=============================================
// File driver.svh
//=============================================

class driver extends
uvm_driver#(my_transaction);
  `uvm_component_utils(driver)

  bit [31:0] mem[bit[31:0]];

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

  // Print the memory contents (addr 0 to 255)
  task dump();
    for (int i = 0; i < 256; i++) begin
      if ((i % 8) == 0 )
        $write("\n%4x: ", i);
      if (mem.exists(i))
        $write("%8x ", mem[i]);
      else
        $write("%8x ", 0);
    end
    $display("");
  endtask

  task run_phase(uvm_phase phase);
    forever begin
      my_transaction t;
      seq_item_port.get_next_item(t);
      `uvm_info("DRVR",
        $sformatf("Got %s", t.convert2string()),
UVM_HIGH)
      if (t.rw == READ) begin
        if (!mem.exists(t.addr))
          t.data = 0;
        else
          t.data = mem[t.addr]; // READ
        #10;
      end
      else if (t.rw == WRITE) begin
        mem[t.addr] = t.data; // WRITE
        // TODO: Error mode. Divisible by 8?
        // if (t.addr[2:0] == 'b000)
        //   mem[t.addr] += 1;
        // write the wrong value.
        #5;
      end
      else if (t.rw == IDLE) begin
        // No delay
      end
      seq_item_port.item_done();
    end
  endtask
endclass
```