```
//Design: synchronous d-flip-flop w/ enable
module my_dut (input clk, reset, d, enable, output q);
 reg qreg;
 always @(posedge clk)
  if (reset) qreg <= 1'b0;
  else if (enable) qreg <= d;
  else qreg <= qreg;
 assign q = qreg;
endmodule

module tb_top;
 import my_pkg::*;
 `include "test_lib.sv"
 bit clk, reset, d, enable, q, qbar;
 my_if vif();
 my_dut dut (.clk(vif.clk), .reset(vif.reset), .d(vif.d),
                        .enable(vif.enable), .q(vif.q));
 always #5 vif.clk = ~vif.clk;
 initial vif.clk = 1'b0;
 initial begin
  uvm_config_db#(virtual my_if)::set(null, "*", "vif", vif);
  run_test("my_base_test");
 end
 initial begin
  $dumpfile("dump.vcd");
  $dumpvars(0, tb_top);
 end
endmodule

class my_base_test extends uvm_test;
 `uvm_component_utils(my_base_test)
 my_env m_env;
 function new(string name="my_base_test", uvm_component parent);
  super.new(name, parent);
 endfunction
 virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  m_env = my_env::type_id::create("m_env", this);
 endfunction
 virtual function void end_of_elaboration_phase(uvm_phase phase);
  super.end_of_elaboration_phase(phase);
  uvm_top.print_topology();
 endfunction
 virtual task run_phase(uvm_phase phase);
  my_sequence m_sequence = my_sequence::type_id::
                          create("m_sequence");
  phase.raise_objection(this);
  m_sequence.start(m_env.m_agent.m_sequencer);
  phase.drop_objection(this);
 endtask
endclass

interface my_if;
 bit reset, clk, d, enable, q;
 m_reset: assert property (@(posedge clk) reset |=> !q);
 property p_enable;
  bit d_prev, q_prev, enable_prev;
  @(posedge clk) (enable & !reset,d_prev = d)|=>(q == d_prev);
 endproperty
 m_enable: assert property (p_enable);
 c_enable: cover property (p_enable);
endinterface

class my_env extends uvm_env;
 `uvm_component_utils(my_env)
 my_agent m_agent;
 my_scoreboard m_scrb;
 my_coverage m_cov;
 uvm_analysis_port #(my_item) m_port;
 function new(string name="my_env", uvm_component parent);
  super.new(name, parent);
 endfunction
 virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  m_agent = my_agent::type_id::create("m_agent", this);
  m_scrb = my_scoreboard::type_id::create("m_scrb", this);
  m_cov = my_coverage::type_id::create("m_cov", this);
  m_port = new("m_port", this);
 endfunction
 virtual function void connect_phase(uvm_phase phase);
  m_agent.m_mon.m_port.connect(m_scrb.m_export);
```

```
  m_agent.m_mon.m_port.connect(m_cov.m_export);
 endfunction
endclass

class my_agent extends uvm_agent;
 `uvm_component_utils(my_agent)
 my_monitor m_mon;
 my_driver m_driver;
 uvm_sequencer #(my_item) m_sequencer;
 function new(string name="my_agent", uvm_component parent);
  super.new(name, parent);
 endfunction
 virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  m_mon = my_monitor::type_id::create("m_mon", this);
  m_driver = my_driver::type_id::create("m_driver", this);
  m_sequencer = uvm_sequencer#(my_item)::type_id::
                                create("m_sequencer", this);
 endfunction
 virtual function void connect_phase(uvm_phase phase);
  super.connect_phase(phase);
  m_driver.seq_item_port.connect(m_sequencer.seq_item_export);
 endfunction
endclass

class my_scoreboard extends uvm_scoreboard;
 `uvm_component_utils(my_scoreboard)
 uvm_analysis_imp #(my_item, my_scoreboard) m_export;
 my_item m_item_prev;
 function new(string name="my_scoreboard",
              uvm_component parent);
  super.new(name, parent);
 endfunction
 virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  m_export = new("m_export", this);
 endfunction
 virtual function void write(my_item m_item);
  if (m_item_prev == null) begin
   m_item_prev = my_item::type_id::create("m_item_prev");
   m_item_prev.do_copy(m_item);
  end
  else begin
   if (m_item.q != m_item_prev.reset? 1'b0 : m_item_prev.enable?
    m_item_prev.d : m_item_prev.q)`uvm_error("SCRB","wrong q")
   m_item_prev.do_copy(m_item);
  end
 endfunction
endclass

class my_coverage extends uvm_subscriber #(my_item);
 `uvm_component_utils(my_coverage)
 uvm_analysis_imp #(my_item, my_coverage) m_export;
 my_item m_item;
 covergroup Cov;
  option.per_instance = 1;
  cp_reset: coverpoint m_item.reset;
  cp_enable: coverpoint m_item.enable;
  cp_d : coverpoint m_item.d;
  cc_all: cross cp_reset, cp_enable, cp_d;
 endgroup
 function new(string name="my_scoreboard",
              uvm_component parent);
  super.new(name, parent);
  Cov = new();
 endfunction
 virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  m_export = new("m_export", this);
 endfunction
 virtual function void write(my_item t);
  this.m_item = t;
  Cov.sample();
 endfunction
 virtual function void report_phase(uvm_phase phase);
  super.report_phase(phase);
  $display("Coverage = %d", Cov.get_coverage());
 endfunction
endclass

class my_item extends uvm_sequence_item;
 `uvm_object_utils(my_item)
 rand logic reset;
 rand logic enable;
```

```
 rand logic d;
 logic q;
 rand int delay;
 constraint c_reset {reset dist {1:=1, 0:=5};}
 constraint c_delay {delay < 3;}
 function new(string name="");
  super.new(name);
 endfunction
 virtual function void do_copy(uvm_object rhs);
  my_item tx;
  $cast(tx, rhs);
  super.do_copy(rhs);
  reset = tx.reset; enable = tx.enable;
  d = tx.d; q = tx.q; delay = tx.delay;
 endfunction
endclass

class my_sequence extends uvm_sequence #(my_item);
 `uvm_object_utils(my_sequence)
 function new(string name="");
  super.new(name);
 endfunction
 virtual task body();
  req = my_item::type_id::create("m_item");
  repeat (50) begin
  if (!req.randomize()) `uvm_error("MYERR","Can't randomize");
   start_item(req);
   finish_item(req);
  end
 endtask
endclass

class my_monitor extends uvm_monitor;
 `uvm_component_utils(my_monitor)
 uvm_analysis_port #(my_item) m_port;
 my_item m_item;
 virtual my_if vif;
 function new(string name="my_monitor", uvm_component parent);
  super.new(name, parent);
 endfunction
 virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  m_port = new("m_port", this);
  if (!uvm_config_db#(virtual my_if)::get(this,"","vif",vif))
    `uvm_fatal("VIF", "can't get vif")
 endfunction
 virtual task run_phase(uvm_phase phase);
  super.run_phase(phase);
  m_item = my_item::type_id::create("m_item");
  forever begin
   @(posedge vif.clk);
   m_item.reset = vif.reset;
   m_item.enable = vif.enable;
   m_item.d = vif.d;
   m_item.q = vif.q;
   m_port.write(m_item);
  end
 endtask
endclass

class my_driver extends uvm_driver #(my_item);
 `uvm_component_utils(my_driver)
 virtual my_if vif;
 function new(string name="my_driver", uvm_component parent);
  super.new(name, parent);
 endfunction
 virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  if (!uvm_config_db#(virtual my_if)::get(this,"","vif",vif))
   `uvm_fatal("VIF", "can't get vif")
 endfunction
 virtual task run_phase(uvm_phase phase);
  super.run_phase(phase);
  forever begin
   @(posedge vif.clk);
   seq_item_port.get_next_item(req);
   repeat (req.delay) @(vif.clk);
    vif.reset = req.reset;
    vif.enable = req.enable;
    vif.d = req.d;
   seq_item_port.item_done();
  end
 endtask
endclass
```