Design

```
 8  module shift_reg (clk, reset, serial_in, direction, mode, datain, dataout);
 9  input clk, reset, serial_in, direction, mode;
10  input [5:0] datain;
11  output reg [5:0] dataout;
12
13  always @(posedge clk or posedge reset) begin
14     if (reset)
15        dataout <= 0;
16     else
17        if (mode) // rotate
18           if (direction) // left
19              dataout <= {datain[4:0], datain[5]};
20           else
21              dataout <= {datain[0], datain[5:1]};
22        else // shift
23           if (direction) // left
24              dataout <= {datain[4:0], serial_in};
25           else
26              dataout <= {serial_in, datain[5:1]};
27  end
28  endmodule
```

Interface

```
 8   interface shift_reg_if (clk);
 9     input clk;
10     logic reset;
11     logic serial_in, direction, mode;
12     logic [5:0] datain, dataout;
13   endinterface : shift_reg_if
```

```
shared_pkg.sv
shift_reg.v
shift_reg_config.sv
shift_reg_seq_item.sv
shift_reg_sequence.sv
shift_reg_sequencer.sv
shift_reg_driver.sv
shift_reg_if.sv
shift_reg_scoreboard.sv
shift_reg_monitor.sv
shift_reg_coverage.sv
shift_reg_agent.sv
shift_reg_env.sv
shift_reg_test.sv
top.sv
```

TOP

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

module top();
  bit clk;
  initial begin
    clk = 0;
    forever
      #1 clk = ~clk;
  end

  shift_reg_if shift_regif (clk);
  shift_reg DUT (clk, shift_regif.reset, shift_regif.serial_in,
  shift_regif.direction, shift_regif.mode, shift_regif.datain, shift_regif.dataout);

  initial begin
    uvm_config_db#(virtual shift_reg_if)::set(null,"uvm_test_top","shift_reg_IF",shift_regif);
    run_test("shift_reg_test");
  end
endmodule
```

ENV

```
package shift_reg_env_pkg;
import uvm_pkg::*;
import shift_reg_agent_pkg::*;
import shift_reg_scoreboard_pkg::*;
import shift_reg_coverage_pkg::*;
`include "uvm_macros.svh"

class shift_reg_env extends uvm_env;
  `uvm_component_utils(shift_reg_env)
  shift_reg_agent agt;
  shift_reg_scoreboard sb;
  shift_reg_coverage cov;

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

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    agt = shift_reg_agent::type_id::create("agt",this);
    sb = shift_reg_scoreboard::type_id::create("sb",this);
    cov = shift_reg_coverage::type_id::create("cov",this);
  endfunction

  function void connect_phase(uvm_phase phase);
    agt.agt_ap.connect(sb.sb_export);
    agt.agt_ap.connect(cov.cov_export);
  endfunction
endclass
endpackage
```

Test

```
package shift_reg_test_pkg;
import shift_reg_env_pkg::*;
import shift_reg_config_pkg::*;
import uvm_pkg::*;
import shift_reg_sequence_pkg::*;
import shift_reg_seq_item_pkg::*;
`include "uvm_macros.svh"
class shift_reg_test extends uvm_test;
  `uvm_component_utils (shift_reg_test)
  shift_reg_env env;
  shift_reg_config shift_reg_cfg;
  shift_reg_main_sequence main_seq;
  shift_reg_reset_sequence reset_seq;
  function new (string name = "shift_reg_test", uvm_component parent = null);
    super.new(name,parent);
  endfunction
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    env = shift_reg_env::type_id::create("env",this);
    shift_reg_cfg = shift_reg_config::type_id::create("shift_reg_cfg");
    main_seq = shift_reg_main_sequence::type_id::create("main_seq");
    reset_seq = shift_reg_reset_sequence::type_id::create("reset_seq");
    if (!uvm_config_db#(virtual shift_reg_if)::get(this,"","shift_reg_IF",shift_reg_cfg.shift_reg_vif))
      `uvm_fatal("build_phase","Test - Unable to get the virtual interface of the shift_reg from the uvm_config_db")
    shift_reg_cfg.sel_mode = UVM_ACTIVE;
    uvm_config_db#(shift_reg_config)::set(this,"*","CFG",shift_reg_cfg);
    factory.set_type_override_by_type(shift_reg_seq_item::get_type(), shift_reg_seq_item_disable_rst::get_type());
  endfunction
  task run_phase(uvm_phase phase);
    super.run_phase(phase);
    phase.raise_objection(this);
    // reset Sequence
    `uvm_info("run_phase","Reset Asserted",UVM_LOW)
    reset_seq.start(env.agt.sqr);
    `uvm_info("run_phase","Reset Deasserted",UVM_LOW)
    // main Sequence
    `uvm_info("run_phase","Stimulus Generation Started",UVM_LOW)
    main_seq.start(env.agt.sqr);
    `uvm_info("run_phase","Stimulus Generation Ended",UVM_LOW)
    phase.drop_objection(this);
  endtask
endclass
endpackage
```

Config

```
package shift_reg_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class shift_reg_config extends uvm_object;
        `uvm_object_utils(shift_reg_config)
        virtual shift_reg_if shift_reg_vif;
        uvm_active_passive_enum sel_mode;
        function new(string name = "shift_reg_config");
            super.new(name);
        endfunction
    endclass
endpackage
```

Driver

```
1   package shift_reg_driver_pkg;
2   import uvm_pkg::*;
3   import shift_reg_config_pkg::*;
4   import shift_reg_seq_item_pkg::*;
5   import shift_reg_sequence_pkg::*;
6   `include "uvm_macros.svh"
7   class shift_reg_driver extends uvm_driver #(shift_reg_seq_item);
8     `uvm_component_utils(shift_reg_driver)
9     virtual shift_reg_if shift_reg_vif;
10    shift_reg_seq_item stim_seq_item;
11
12    function new (string name ="shift_reg_driver",uvm_component parent = null);
13      super.new(name,parent);
14    endfunction
15
16    task run_phase (uvm_phase phase);
17      super.run_phase(phase);
18      forever begin
19        stim_seq_item = shift_reg_seq_item::type_id::create("stim_seq_item");
20        seq_item_port.get_next_item(stim_seq_item);
21        shift_reg_vif.direction = stim_seq_item.direction;
22        shift_reg_vif.mode = stim_seq_item.mode;
23        shift_reg_vif.datain = stim_seq_item.datain;
24        shift_reg_vif.serial_in = stim_seq_item.serial_in;
25        shift_reg_vif.reset = stim_seq_item.reset;
26        @(negedge shift_reg_vif.clk);
27        seq_item_port.item_done();
28        `uvm_info("run_phase",stim_seq_item.convert2string_stimulus(),UVM_HIGH)
29      end
30    endtask
31  endclass
32  endpackage
```

Sequencer

```
1   package shift_reg_sequencer_pkg;
2   import uvm_pkg::*;
3   import shift_reg_seq_item_pkg::*;
4   `include "uvm_macros.svh"
5
6   class shift_reg_sequencer extends uvm_sequencer #(shift_reg_seq_item);
7       `uvm_component_utils(shift_reg_sequencer);
8
9       function new(string name = "shift_reg_sequencer", uvm_component parent = null);
10          super.new(name,parent);
11      endfunction
12  endclass
13  endpackage
```

Sequence

```
1   package shift_reg_sequence_pkg;
2
3   import uvm_pkg::*;
4   import shift_reg_seq_item_pkg::*;
5   import shared_pkg::*;
6   `include "uvm_macros.svh"
7
8   class shift_reg_main_sequence extends uvm_sequence #(shift_reg_seq_item);
9       `uvm_object_utils(shift_reg_main_sequence);
10      shift_reg_seq_item seq_item;
11      function new(string name = "shift_reg_main_sequence");
12          super.new(name);
13      endfunction
14      task body ();
15          repeat(10000) begin
16              seq_item = shift_reg_seq_item::type_id::create("seq_item");
17              start_item(seq_item);
18              assert(seq_item.randomize());
19              finish_item(seq_item);
20          end
21      endtask
22  endclass
23
24  class shift_reg_reset_sequence extends uvm_sequence #(shift_reg_seq_item);
25      `uvm_object_utils(shift_reg_reset_sequence);
26      shift_reg_seq_item seq_item;
27
28      function new(string name = "shift_reg_reset_sequence");
29          super.new(name);
30      endfunction
31      task body ();
32          repeat(10000) begin
33              seq_item = shift_reg_seq_item::type_id::create("seq_item");
34              start_item(seq_item);
35              seq_item.reset = 1;
36              seq_item.serial_in = 0;
37              seq_item.direction = direction_e'(0);
38              seq_item.mode = mode_e'(0);
39              seq_item.datain = 0;
40              finish_item(seq_item);
41          end
42      endtask
43  endclass
44  endpackage
```

Sequence item

```
1   package shift_reg_seq_item_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   class shift_reg_seq_item extends uvm_sequence_item;
5       `uvm_object_utils(shift_reg_seq_item);
6
7       rand bit serial_in;
8       rand bit direction;
9       rand bit mode;
10      rand bit [5:0]datain;
11      rand bit reset;
12      bit [5:0] dataout;
13
14      function new(string name = "shift_reg_seq_item");
15          super.new(name);
16      endfunction
17
18      function string convert2string();
19          return $sformatf ("%s diretion=%0b,mode=%0b,serial_in=%0b,datain=%0d,dataout=%0d",
20              super.convert2string(),direction,mode,serial_in,datain,dataout);
21      endfunction
22
23      function string convert2string_stimulus();
24          return $sformatf ("%s diretion=%0b,mode=%0b,serial_in=%0b,datain=%0d",
25              super.convert2string(),direction,mode,serial_in,datain);
26      endfunction
27
28      constraint rst_con {
29          reset dist {1 :/ 2, 0 :/ 98};
30      }
31  endclass
32
33  class shift_reg_seq_item_disable_rst extends shift_reg_seq_item;
34      `uvm_object_utils(shift_reg_seq_item_disable_rst);
35      function new(string name = "shift_reg_seq_item");
36          super.new(name);
37      endfunction
38
39      constraint rst_c {
40          reset == 0;
41      }
42  endclass
43  endpackage
```

Agent

```
1   package shift_reg_agent_pkg;
2   import uvm_pkg::*;
3   import shift_reg_config_pkg::*;
4   import shift_reg_driver_pkg::*;
5   import shift_reg_monitor_pkg::*;
6   import shift_reg_sequencer_pkg::*;
7   import shift_reg_seq_item_pkg::*;
8   import shift_reg_sequence_pkg::*;
9   `include "uvm_macros.svh"
10  class shift_reg_agent extends uvm_agent;
11  `uvm_component_utils(shift_reg_agent)
12  shift_reg_sequencer sqr;
13  shift_reg_monitor mon;
14  shift_reg_driver drv;
15  shift_reg_config shift_reg_cfg;
16  uvm_analysis_port #(shift_reg_seq_item) agt_ap;
17
18  function new (string name = "shift_reg_agent", uvm_component parent = null);
19      super.new(name,parent);
20  endfunction
21  function void build_phase(uvm_phase phase);
22      super.build_phase(phase);
23      if (!uvm_config_db #(shift_reg_config)::get(this, "", "CFG",shift_reg_cfg)) begin
24          `uvm_fatal("build_phase","Unable to get configuration object")
25      end
26      if (shift_reg_cfg.sel_mode == UVM_ACTIVE) begin
27          sqr = shift_reg_sequencer::type_id::create("sqr",this);
28          drv = shift_reg_driver::type_id::create("drv",this);
29      end
30      mon = shift_reg_monitor::type_id::create("mon",this);
31      agt_ap = new("agt_ap",this);
32  endfunction
33  function void connect_phase(uvm_phase phase);
34      super.connect_phase(phase);
35      if (shift_reg_cfg.sel_mode == UVM_ACTIVE) begin
36          drv.shift_reg_vif = shift_reg_cfg.shift_reg_vif;
37          drv.seq_item_port.connect(sqr.seq_item_export);
38      end
39      mon.shift_reg_vif = shift_reg_cfg.shift_reg_vif;
40      mon.mon_ap.connect(agt_ap);
41  endfunction
42  endclass
43  endpackage
```

Monitor

```
package shift_reg_monitor_pkg;
import uvm_pkg::*;
import shift_reg_config_pkg::*;
import shift_reg_seq_item_pkg::*;
import shared_pkg::*;
`include "uvm_macros.svh"
class shift_reg_monitor extends uvm_monitor;
    `uvm_component_utils(shift_reg_monitor)
    virtual shift_reg_if shift_reg_vif;
    shift_reg_seq_item rsp_seq_item;
    uvm_analysis_port #(shift_reg_seq_item) mon_ap;

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

    function void build_phase (uvm_phase phase);
        super.build_phase(phase);
        mon_ap = new("mon_ap",this);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            rsp_seq_item = shift_reg_seq_item::type_id::create("rsp_seq_item");
            @(negedge shift_reg_vif.clk);
            rsp_seq_item.direction = direction_e'(shift_reg_vif.direction);
            rsp_seq_item.mode = mode_e'(shift_reg_vif.mode);
            rsp_seq_item.datain = shift_reg_vif.datain;
            rsp_seq_item.serial_in = shift_reg_vif.serial_in;
            rsp_seq_item.reset = shift_reg_vif.reset;
            rsp_seq_item.dataout = shift_reg_vif.dataout;
            mon_ap.write(rsp_seq_item);
            `uvm_info("run_phase",rsp_seq_item.convert2string(),UVM_HIGH)
        end
    endtask
endclass

endpackage
```

Shared PKG

```
package shared_pkg;
    typedef enum { SHIFT, ROTATE } mode_e;
    typedef enum { RIGHT, LEFT } direction_e;
endpackage
```

Scoreboard

```
package shift_reg_scoreboard_pkg;
import uvm_pkg::*;
import shared_pkg::*;
import shift_reg_seq_item_pkg::*;
`include "uvm_macros.svh"
class shift_reg_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(shift_reg_scoreboard);
    uvm_analysis_export #(shift_reg_seq_item) sb_export;
    uvm_tlm_analysis_fifo #(shift_reg_seq_item) sb_fifo;
    shift_reg_seq_item seq_item_sb;
    logic [5:0] shift_reg_out_ref;
    int error_count=0;
    int correct_count=0;
    function new(string name ="shift_reg_scoreboard",uvm_component parent = null);
        super.new(name,parent);
    endfunction
    function void build_phase (uvm_phase phase);
        super.build_phase(phase);
        sb_export=new("sb_export",this);
        sb_fifo=new("sb_fifo",this);
    endfunction
    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        sb_export.connect(sb_fifo.analysis_export);
    endfunction
    task run_phase (uvm_phase phase);
        super.run_phase(phase);
            forever begin
                sb_fifo.get(seq_item_sb);

                if (seq_item_sb.dataout != shift_reg_out_ref) begin
                    `uvm_error("run_phase",$sformatf("Comparison failed, Transaction recieved by the DUT:%s while the refecrence out:0b%0b"
                                                     ,seq_item_sb.convert2string(),shift_reg_out_ref));
                    error_count++;
                end
                else begin
                    `uvm_info("run_phase",$sformatf("correct shift_reg_out :%s ",seq_item_sb.convert2string()),UVM_HIGH);
                    correct_count++;
                end
            end
    endtask
```

```
    task ref_model (shift_reg_seq_item seq_item_chk);
        if (seq_item_chk.reset)
            shift_reg_out_ref = 0;
        else begin
            case (seq_item_chk.mode)
                SHIFT  : begin
                    if (seq_item_chk.direction == LEFT)
                        shift_reg_out_ref = {seq_item_chk.datain[4:0],seq_item_chk.serial_in};
                    else if (seq_item_chk.direction == RIGHT) begin
                        shift_reg_out_ref  = {seq_item_chk.serial_in,seq_item_chk.datain[5:1]};
                    end
                end
                ROTATE : begin
                    if (seq_item_chk.direction == LEFT)
                        shift_reg_out_ref = {seq_item_chk.datain[4:0],seq_item_chk.datain[5]};
                    else if (seq_item_chk.direction == RIGHT) begin
                        shift_reg_out_ref  = {seq_item_chk.datain[0],seq_item_chk.datain[5:1]};
                    end
                end
            endcase
        end
    endtask
    function void report_phase (uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("report_phase",$sformatf("Total successful counts :%0d",correct_count),UVM_MEDIUM);
        `uvm_info("report_phase",$sformatf("Total failed counts :%0d",error_count),UVM_MEDIUM);
    endfunction
endclass
endpackage
```

Do File

```
1   vlib work
2   vlog -f src_files.list +cover -covercells
3   vsim -voptargs=+acc work.top -cover -classdebug -uvmcontrol=all
4   add wave /top/shift_regif/*
5   run 0
6   coverage save top.ucdb -onexit
7   run -all
8   # quit -sim
9   # vcover report top.ucdb -details -annotate -all -output cov_report.txt
```