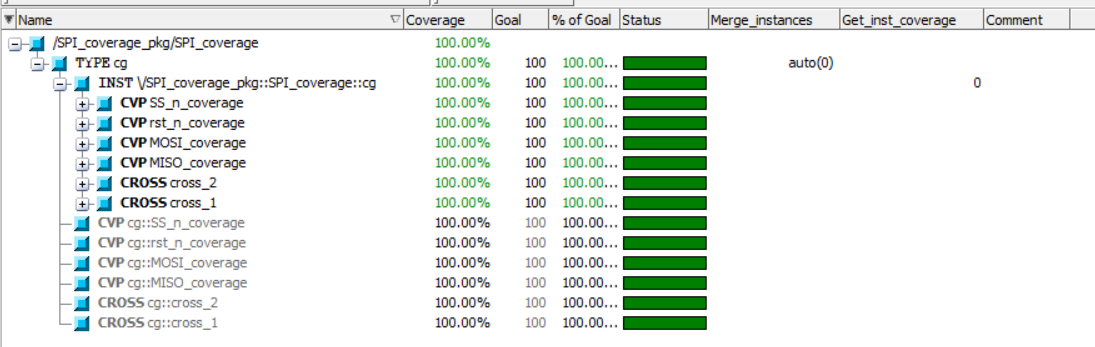
  
  
A screen shot of a computer

Description automatically generated  
  
  


Golden\_wrapper

module Master(SS\_n,MOSI,MISO,clk,rst\_n);

input MOSI,SS\_n,clk,rst\_n;

 output MISO;

 wire rx\_valid;

wire [9:0] rx\_data;

wire [7:0] tx\_data;

wire tx\_valid;

 SPI   s1 (clk,rst\_n,MOSI,MISO,SS\_n,rx\_data,rx\_valid,tx\_data,tx\_valid);

RAM\_pr2  R1(clk,rst\_n,rx\_data,rx\_valid,tx\_data,tx\_valid);

endmodule

GOLDEN\_RAM

module RAM\_pr2(clk,rst\_n,din,rx\_valid,dout,tx\_valid);

parameter MEM\_DEPTH = 256;

parameter ADDR\_SIZE = 8;

input [9:0] din;

input rx\_valid,clk,rst\_n;

output reg[7:0] dout;

output reg tx\_valid;

reg [ADDR\_SIZE-1:0] wr\_address,rd\_address;

reg [ADDR\_SIZE-1:0] mem\_1 [MEM\_DEPTH-1:0];

always @(posedge clk) begin

    if(!rst\_n)

    begin

    dout<=0;

    tx\_valid<=0;

    end

    else

    begin

    if(rx\_valid==1)

    begin

    case (din[9:8])

    2'b00: begin

      wr\_address<= din[7:0];

      tx\_valid<=0;

    end

    2'b01: begin

        mem\_1[wr\_address]<= din[7:0];

        tx\_valid<=0;

    end

    2'b10:begin

         rd\_address<= din[7:0];

         tx\_valid<=0;

    end

    2'b11: begin

        dout<= mem\_1[rd\_address];

        tx\_valid<=1;

    end

     default: wr\_address<= din[7:0];

    endcase

    end

end

end

endmodule

GOLDEN\_SPI

module SPI (clk,rst\_n,MOSI,MISO,SS\_n,rx\_data,rx\_valid,tx\_data,tx\_valid);

parameter IDLE=3'b000;

parameter CHK\_CMD=3'b001;

parameter WRITE=3'b010;

parameter READ\_ADDRESS=3'b011;

parameter READ\_DATA=3'b100;

input MOSI,SS\_n,clk,rst\_n;

 output  rx\_valid;

output reg [9:0] rx\_data;

input [7:0] tx\_data;

input tx\_valid;

reg  rd\_address\_first;

output reg MISO;

reg[3:0] count\_1;

reg [3:0]count\_2;

reg [3:0] count\_3;

(\* fsm\_encoding = "one\_hot" \*)

reg [2:0] ns,cs;

always@(cs,MOSI,SS\_n)

begin

case (cs)

    IDLE:begin

        if(!SS\_n)

        ns=CHK\_CMD;

        else

        ns= IDLE;

    end

    CHK\_CMD:begin

        if(SS\_n==0)

        begin

        if(MOSI==0)

        ns=WRITE;

        else

        if(rd\_address\_first==0)

        begin

        ns=READ\_ADDRESS;

        end

        else

        begin

        ns=READ\_DATA;

        end

        end

        else

        ns= IDLE;

    end

    WRITE:begin

        if(SS\_n)

        ns= IDLE;

        else

        begin

        if(count\_1 <10)

        ns= WRITE;

        end

    end

    READ\_ADDRESS:begin

        if(SS\_n)

        ns=IDLE;

        else

        begin

            if(count\_2 < 10)

            ns=READ\_ADDRESS;

    end

    end

    READ\_DATA: begin

         if(SS\_n)

        ns=IDLE;

        else

        begin

            if(count\_3 < 8)

            ns=READ\_DATA;

    end

    end

    default: ns= IDLE;

endcase

end

always@(posedge clk or negedge rst\_n)

begin

  if(!rst\_n)

   cs<=IDLE;

else

cs<= ns;

end

always@(posedge clk or negedge rst\_n)

begin

if(!rst\_n)

begin

  rx\_data<=0;

  // rx\_valid<=0;

  count\_1<=0;

   count\_2<=0;

  rd\_address\_first<=0;

  count\_3<=0;

/\* MISO<=0;\*/

end

else

case(cs)

IDLE:begin

  /\*rx\_valid<=0;\*/

  count\_1<=0;

 count\_2<=0;

  count\_3<=0;

end

CHK\_CMD:begin

    // rx\_valid<=0;

    count\_1<=0;

   count\_2<=0;

    count\_3<=0;

end

WRITE: begin

    if(count\_1<10)

    begin

        rx\_data<={rx\_data[8:0],MOSI};

        count\_1<=count\_1+1;

    end

       /\* else

        rx\_valid<=1;\*/

    end

READ\_ADDRESS: begin

 rd\_address\_first<=1;

    if(count\_2<10)

    begin

        rx\_data<={rx\_data[8:0],MOSI };

       count\_2<=count\_2+1;

    end

    /\*    else

        rx\_valid<=1;\*/

    end

    READ\_DATA: begin

    rd\_address\_first<=0;

    if(count\_2<10)

    begin

        rx\_data<={rx\_data[8:0],MOSI };

       count\_2<=count\_2+1;

    end

        else

        begin

       /\* rx\_valid<=1;\*/

        if(count\_3 < 8 )

        begin

            if(tx\_valid==1)

            begin

            MISO <= tx\_data[7-count\_3];

            count\_3<=count\_3+1;

            end

        end

        end

    end

    default:begin

  /\* rx\_valid<=0;\*/

  count\_1<=0;

 count\_2<=0;

  count\_3<=7;

end

    endcase

end

assign rx\_valid= ((count\_2==10 || count\_1==10) && cs!=IDLE)? 1:0;

endmodule

DUT\_wrapper

module SPI\_Wrapper(MISO, MOSI,SS\_n,rst\_n,clk);

input MOSI, SS\_n,clk,rst\_n;

output MISO;

wire [9:0] rxdata;

wire [7:0] txdata;

wire rx\_valid,tx\_valid;

//instantiation of spi slave and ram

SPI\_RAM #(256,8) R (rxdata, txdata, rx\_valid, tx\_valid , clk, rst\_n);

SIP\_SLAVE  S (MOSI,MISO,SS\_n,clk,rst\_n,rxdata,rx\_valid, txdata, tx\_valid);

endmodule

DUT\_SPI

module SIP\_SLAVE (MOSI,MISO,SS\_n,clk,rst\_n,rx\_data,rx\_valid, tx\_data, tx\_valid);

input MOSI,SS\_n,clk,rst\_n,tx\_valid;

input [7:0] tx\_data;

output reg MISO;

output rx\_valid;

output reg [9:0] rx\_data;

//states

parameter IDLE=3'b000;

parameter READ\_DATA=3'b001;

parameter READ\_ADD=3'b011;

parameter CHK\_CMD=3'b111;

parameter WRITE=3'b100;

reg [2:0] cs,ns;

reg rd\_flag; // to check if READ\_ADD state has been executed or not (high if executed)

reg [3:0] state\_countout;

reg[2:0] MISO\_CountOut;

wire counter\_enable; // to count 10 clock cycles while recieving data

wire counter\_done; // flag sent by the counter, high when 10 cycles are completed

wire MISO\_CountEn;

reg [2:0] count\_once=0;

reg read\_once=0;

//state logic

always@(posedge clk or negedge rst\_n) begin

  if(~rst\_n) begin

    cs<= IDLE;

  end

  else begin

    cs<=ns;

  end

end

//next state logic

always@(\*) begin

  case(cs)

  IDLE: begin

    if(~SS\_n)

      ns=CHK\_CMD;

    else

      ns=IDLE;

  end

  CHK\_CMD:begin

    if(~SS\_n) begin

      if(MOSI && ~rd\_flag) begin

         ns=READ\_ADD;

        end

        else if(MOSI && rd\_flag) begin

         ns=READ\_DATA;

        end

        else  begin

         ns=WRITE;

        end

    end

    else

      ns=IDLE;

    end

  READ\_DATA:

    if(SS\_n) begin

      ns=IDLE;

    end

    else begin

      ns=READ\_DATA;

    end

  READ\_ADD:

    if(SS\_n) begin

      ns=IDLE;

    end

    else begin

      ns=READ\_ADD;

    end

  WRITE:

     if(SS\_n) begin

      ns=IDLE;

    end

    else begin

      ns=WRITE;

    end

    endcase

end

// output logic

always@ (posedge clk or negedge rst\_n) begin

  if(~rst\_n) begin

    rd\_flag <= 0;

  end

  if(counter\_enable) begin         //

    rx\_data<={rx\_data[8:0], MOSI};

  end

  if(cs==READ\_ADD && counter\_done) begin

    rd\_flag<=1;

  end

  else if (cs==READ\_DATA && counter\_done)begin

    rd\_flag<=0;

  end

  if(MISO\_CountEn) begin

    MISO<= tx\_data[MISO\_CountOut];

  end

end

// counters logic

always@(posedge clk or negedge rst\_n) begin

  if(~rst\_n) begin

    state\_countout<=0;

    MISO\_CountOut<=7; // counter\_down, to send MSB firstly to master

  end

  else begin

    if(ns == IDLE) begin // else if

      state\_countout<=0;

      read\_once<=0;

    end

    else if(counter\_enable) begin

      state\_countout<=state\_countout+1;

    end

    if(MISO\_CountEn) begin

      MISO\_CountOut<=MISO\_CountOut-1;

      count\_once<=count\_once+1;

    end

     if(count\_once==7)

      begin

      read\_once<=1;

      count\_once<=0;

    end

  end

end

assign counter\_enable=(cs != IDLE && cs!=CHK\_CMD && ~counter\_done)?1:0;

assign counter\_done = (state\_countout==4'b1010 )?1:0;

assign MISO\_CountEn=(tx\_valid && cs == 3'b001 && ns != IDLE && !read\_once)?1:0;

assign rx\_valid=(counter\_done)? 1:0;

endmodule

DUT\_RAM

module SPI\_RAM (din, dout ,rx\_valid, tx\_valid , clk, rst\_n);

parameter MEM\_DEPTH=256;

parameter ADDR\_SIZE=8;

input clk, rst\_n, rx\_valid;

input [9:0] din;

output  reg tx\_valid;

output reg [ADDR\_SIZE-1:0] dout;

reg [ADDR\_SIZE-1:0] write\_address, read\_address; //buses to hold write or read addresses

reg [ADDR\_SIZE-1:0] memory [MEM\_DEPTH-1:0];

always @(posedge clk or negedge rst\_n) begin

    if (~rst\_n) begin

        dout<=0;

        tx\_valid<=0;

    end

    else if (rx\_valid) begin

        case ( din [9:8])

          2'b00: begin

            write\_address<= din[ADDR\_SIZE-1:0];

            tx\_valid <= 0;

          end

          2'b01: begin

            memory[write\_address] <= din [ADDR\_SIZE-1:0];

            tx\_valid <= 0;

          end

          2'b10: begin

            read\_address<= din [ADDR\_SIZE-1:0];

            tx\_valid <= 0;

          end

          2'b11: begin

            dout<= memory[read\_address];

            tx\_valid<=1;

          end

        endcase

    end

end

endmodule

interface

interface SPI\_if(clk);

    input clk;

    logic MOSI, SS\_n,rst\_n,MISO,MISO\_G;

endinterface : SPI\_if

TOP

import SPI\_test\_pkg::\*;

import uvm\_pkg::\*;

`include "uvm\_macros.svh"

module top();

    bit clk;

    initial begin

        forever

        #1 clk = ~clk;

    end

    SPI\_if SPIif(clk);

    SPI\_Wrapper DUT(SPIif.MISO, SPIif.MOSI,SPIif.SS\_n,SPIif.rst\_n,clk);

    Master GOLDEN(SPIif.SS\_n,SPIif.MOSI,SPIif.MISO\_G,SPIif.clk,rst\_n);

    initial begin

        uvm\_config\_db#(virtual SPI\_if)::set(null, "uvm\_test\_top", "SPI\_IF",SPIif);

        run\_test("SPI\_test");

    end

endmodule : top

TEST

package SPI\_test\_pkg;

    import SPI\_env\_pkg::\*;

    import pkg::\*;

    import SPI\_reset\_seq\_pkg::\*;

    import SPI\_main\_seq\_pkg::\*;

    import uvm\_pkg::\*;

    `include "uvm\_macros.svh"

    class SPI\_test extends  uvm\_test;

        `uvm\_component\_utils(SPI\_test)

        SPI\_env env;

        SPI\_config\_obj SPI\_config\_obj\_test;

        virtual SPI\_if  SPI\_test\_vif;

        SPI\_main\_sequence main\_seq;

        SPI\_reset\_sequence reset\_seq;

        function new(string name = "SPI\_test", uvm\_component parent = null);

            super.new(name,parent);

        endfunction

        function void build\_phase(uvm\_phase phase);

            super.build\_phase(phase);

            env = SPI\_env::type\_id::create("env",this);

            SPI\_config\_obj\_test = SPI\_config\_obj::type\_id::create("SPI\_config\_obj\_test",this);

            main\_seq = SPI\_main\_sequence::type\_id::create("main\_seq",this);

            reset\_seq = SPI\_reset\_sequence::type\_id::create("reset\_seq",this);

            uvm\_config\_db#(virtual SPI\_if)::get(this, "", "SPI\_IF",SPI\_config\_obj\_test.SPI\_config\_vif);

            uvm\_config\_db#(SPI\_config\_obj)::set(this, "\*", "SPI\_IF1",SPI\_config\_obj\_test);

            endfunction

        task run\_phase(uvm\_phase phase);

            super.run\_phase(phase);

            phase.raise\_objection(this);

            `uvm\_info("run\_phase","reset\_asserted",UVM\_LOW)

            reset\_seq.start(env.agt.sqr);

            `uvm\_info("run\_phase","reset\_deasserted",UVM\_LOW)

            `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 : run\_phase

    endclass

endpackage

Environment

package SPI\_env\_pkg;

    import SPI\_agent\_pkg::\*;

    import SPI\_scoreboard\_pkg::\*;

    import SPI\_coverage\_pkg::\*;

    import uvm\_pkg::\*;

    `include "uvm\_macros.svh"

    class SPI\_env extends  uvm\_env;

        `uvm\_component\_utils(SPI\_env)

        SPI\_agent agt;

        SPI\_scoreboard sb;

        SPI\_coverage cov;

        function new(string name ="SPI\_env",uvm\_component parent = null);

            super.new(name,parent);

        endfunction

        function void build\_phase(uvm\_phase phase);

            super.build\_phase(phase);

            agt = SPI\_agent::type\_id::create("agt",this);

            sb = SPI\_scoreboard::type\_id::create("sb",this);

            cov = SPI\_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

Agent

package SPI\_agent\_pkg;

    import uvm\_pkg::\*;

    import my\_sequencer\_pkg::\*;

    import SPI\_driver\_pkg::\*;

    import SPI\_monitor\_pkg::\*;

    import SPI\_seq\_item\_pkg::\*;

    import pkg::\*;

    `include "uvm\_macros.svh"

    class SPI\_agent extends  uvm\_agent;

        `uvm\_component\_utils(SPI\_agent)

        MySequencer sqr;

        SPI\_driver drv;

        SPI\_monitor mon;

        SPI\_config\_obj SPI\_cfg;

        uvm\_analysis\_port #(SPI\_seq\_item) agt\_ap;

        function new(string name = "SPI\_agent",uvm\_component parent = null);

            super.new(name,parent);

        endfunction

        function void build\_phase(uvm\_phase phase);

            super.build\_phase(phase);

            uvm\_config\_db#(SPI\_config\_obj)::get(this, "", "SPI\_IF1",SPI\_cfg);

            sqr = MySequencer::type\_id::create("sqr",this);

            drv = SPI\_driver::type\_id::create("drv",this);

            mon = SPI\_monitor::type\_id::create("mon",this);

            agt\_ap = new("agt\_ap",this);

        endfunction

        function void connect\_phase(uvm\_phase phase);

            drv.SPI\_driver\_vif = SPI\_cfg.SPI\_config\_vif;

            mon.SPI\_monitor\_vif = SPI\_cfg.SPI\_config\_vif;

            drv.seq\_item\_port.connect(sqr.seq\_item\_export);

            mon.mon\_ap.connect(agt\_ap);

        endfunction

    endclass

endpackage

driver

package SPI\_driver\_pkg;

    import uvm\_pkg::\*;

    import SPI\_seq\_item\_pkg::\*;

    `include "uvm\_macros.svh"

    class SPI\_driver extends  uvm\_driver #(SPI\_seq\_item);

        `uvm\_component\_utils(SPI\_driver)

        virtual SPI\_if SPI\_driver\_vif;

        SPI\_seq\_item stim\_seq\_item;

        function new(string name ="SPI\_driver",uvm\_component parent = null);

            super.new(name,parent);

        endfunction

        task run\_phase(uvm\_phase phase);

            super.run\_phase(phase);

            forever begin

                stim\_seq\_item = SPI\_seq\_item::type\_id::create("stim\_seq\_item");

                seq\_item\_port.get\_next\_item(stim\_seq\_item);

                SPI\_driver\_vif.MOSI = stim\_seq\_item.MOSI;

                SPI\_driver\_vif.SS\_n = stim\_seq\_item.SS\_n;

                SPI\_driver\_vif.rst\_n = stim\_seq\_item.rst\_n;

                @(negedge SPI\_driver\_vif.clk);

                stim\_seq\_item.MOSI = SPI\_driver\_vif.MOSI;

                stim\_seq\_item.MISO\_G = SPI\_driver\_vif.MISO\_G;

                seq\_item\_port.item\_done();

                `uvm\_info("run\_phase",stim\_seq\_item.convert2string\_stimulus(),UVM\_HIGH)

            end

        endtask : run\_phase

    endclass : SPI\_driver

endpackage : SPI\_driver\_pkg

sequencer

package my\_sequencer\_pkg;

    import uvm\_pkg::\*;

    import SPI\_seq\_item\_pkg::\*;

    `include "uvm\_macros.svh"

    class MySequencer extends  uvm\_sequencer #(SPI\_seq\_item);

        `uvm\_component\_utils(MySequencer)

        function new(string name = "MySequencer",uvm\_component parent = null);

            super.new(name,parent);

        endfunction

    endclass

endpackage

seq\_item

package SPI\_seq\_item\_pkg;

    import uvm\_pkg::\*;

    `include "uvm\_macros.svh"

    class SPI\_seq\_item extends uvm\_sequence\_item;

        `uvm\_object\_utils(SPI\_seq\_item)

        rand logic SS\_n,rst\_n,MOSI;

       // logic  WRITE\_ADD  = 1;

        //logic READ\_ADD = 1;

        rand logic [12:0] din;

        logic MISO,MISO\_G;

        function new(string name = "SPI\_seq\_item");

            super.new(name);

        endfunction

        function string convert2string();

            return $sformatf("%s rst\_n = 0b%0b,MOSI = 0b%0b,SS\_n = 0b%0b,MISO = 0b%0b",super.convert2string(),rst\_n,MOSI,SS\_n,MISO);

        endfunction

        function string convert2string\_stimulus();

        return $sformatf("rst\_n = 0b%0b,MOSI = 0b%0b,SS\_n = 0b%0b",rst\_n,MOSI,SS\_n);

        endfunction

        constraint c1 {

            rst\_n dist {1:=98,0:=2};

        }

        constraint c2 {

            SS\_n dist {1:=98,0:=2};

        }

        constraint c3 {

            SS\_n dist {0:=98,1:=2};

        }

        constraint c4 {

            din[11:8] == 4'b0000;

        }

        constraint c5 {

            din[11:8] == 4'b0001;

        }

        constraint c6 {

            din[11:8] == 4'b1111;

        }

        constraint c7 {

            din[11:8] == 4'b1110;

        }

    endclass

endpackage

reset\_sequence

package SPI\_reset\_seq\_pkg;

    import SPI\_seq\_item\_pkg::\*;

    import uvm\_pkg::\*;

    `include "uvm\_macros.svh"

    class SPI\_reset\_sequence extends  uvm\_sequence #(SPI\_seq\_item);

        `uvm\_object\_utils(SPI\_reset\_sequence)

        SPI\_seq\_item seq\_item;

        function new(string name = "SPI\_reset\_sequence");

            super.new(name);

        endfunction

        task body;

            seq\_item = SPI\_seq\_item::type\_id::create("seq\_item");

            start\_item(seq\_item);

            seq\_item.rst\_n = 0;

            seq\_item.MOSI = 0;

            seq\_item.SS\_n = 1;

            finish\_item(seq\_item);

        endtask

    endclass

endpackage

main\_sequence

package SPI\_main\_seq\_pkg;

    import uvm\_pkg::\*;

    import SPI\_seq\_item\_pkg::\*;

    `include "uvm\_macros.svh"

    typedef enum logic [2:0] {IDLE=0,CHK=7,WRITE=4,READ\_ADD=3,READ\_DATA=1} e\_CS;

    class SPI\_main\_sequence extends  uvm\_sequence #(SPI\_seq\_item);

        `uvm\_object\_utils(SPI\_main\_sequence)

        SPI\_seq\_item seq\_item;

        logic[12:0] din\_r;

        bit WRITE\_DATA,READ\_DATA;

        bit WRITE = 1;

        int i = 11;

        function new(string name = "SPI\_reset\_sequence");

            super.new(name);

        endfunction

        task body;

            seq\_item = SPI\_seq\_item::type\_id::create("seq\_item");

            repeat(200000) begin

            start\_item(seq\_item);

            if(i > -1) begin

                seq\_item.c2.constraint\_mode(0);

                seq\_item.c3.constraint\_mode(1);

            end

            else begin

                if (READ\_DATA == 0 ) begin

                    seq\_item.c2.constraint\_mode(1);

                    seq\_item.c3.constraint\_mode(0);

                end

                else if (READ\_DATA) begin

                    seq\_item.c2.constraint\_mode(0);

                    seq\_item.c3.constraint\_mode(1);

                    if (i == -12) begin

                    seq\_item.c2.constraint\_mode(1);

                    seq\_item.c3.constraint\_mode(0);

                    end

                end

            end

            if (i == -1) begin

                if (din\_r[9:8] == 2'b00) begin

                    WRITE\_DATA = 1;

                end

                else if (din\_r[9:8] == 2'b01) begin

                    WRITE\_DATA = 0;

                    WRITE = ~ WRITE;

                end

                else if (din\_r[9:8] == 2'b10) begin

                    READ\_DATA = 1;

                end

            end

            else if (seq\_item.rst\_n == 0) begin

                READ\_DATA = 0;

                WRITE = ~WRITE;

            end

            else if (i==-12) begin

                if (din\_r[9:8] == 2'b11) begin

                    READ\_DATA = 0;

                    WRITE = ~ WRITE;

                end

            end

            if (!WRITE\_DATA && WRITE) begin

                seq\_item.c4.constraint\_mode(1);

                seq\_item.c5.constraint\_mode(0);

                seq\_item.c6.constraint\_mode(0);

                seq\_item.c7.constraint\_mode(0);

            end

            else if (!READ\_DATA && !WRITE) begin

                seq\_item.c4.constraint\_mode(0);

                seq\_item.c5.constraint\_mode(0);

                seq\_item.c6.constraint\_mode(0);

                seq\_item.c7.constraint\_mode(1);

            end

            else if (WRITE\_DATA && WRITE) begin

                seq\_item.c5.constraint\_mode(1);

                seq\_item.c4.constraint\_mode(0);

                seq\_item.c6.constraint\_mode(0);

                seq\_item.c7.constraint\_mode(0);

            end

            else if (READ\_DATA && !WRITE) begin

                seq\_item.c6.constraint\_mode(1);

                seq\_item.c5.constraint\_mode(0);

                seq\_item.c4.constraint\_mode(0);

                seq\_item.c7.constraint\_mode(0);

            end

            assert(seq\_item.randomize());

            if (i==11) begin

                din\_r = seq\_item.din;

            end

            if (seq\_item.SS\_n) begin

                seq\_item.MOSI = din\_r[12];

                i = 11;

            end

            else begin

                seq\_item.MOSI = din\_r[i];

                if(i<=-1) begin

                    seq\_item.MOSI = din\_r[0];

                end

                if (seq\_item.rst\_n) begin

                    i = i-1;

                end

                else begin

                    i = 11;

                end

            end

            finish\_item(seq\_item);

        end

        endtask

    endclass

endpackage

configuration object

package pkg;

    import uvm\_pkg::\*;

    `include "uvm\_macros.svh"

    class SPI\_config\_obj extends  uvm\_object;

        `uvm\_object\_utils(SPI\_config\_obj)

        virtual SPI\_if SPI\_config\_vif;

        function new(string name = "SPI\_config\_obj");

            super.new(name);

        endfunction

    endclass

endpackage

scoreboard

package SPI\_scoreboard\_pkg;

    import uvm\_pkg::\*;

    import SPI\_seq\_item\_pkg::\*;

    `include "uvm\_macros.svh"

    class SPI\_scoreboard extends uvm\_scoreboard;

        `uvm\_component\_utils(SPI\_scoreboard)

        uvm\_analysis\_export #(SPI\_seq\_item) sb\_export;

        uvm\_tlm\_analysis\_fifo #(SPI\_seq\_item) sb\_fifo;

        SPI\_seq\_item seq\_item\_sb;

        logic [5:0] dataout\_ref;

        int error\_count = 0;

        int correct\_count = 0;

        function new(string name = "SPI\_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);

                //ref\_model(seq\_item\_sb);

                if(seq\_item\_sb.MISO != seq\_item\_sb.MISO\_G) begin

                    `uvm\_error("run\_phase",$sformatf("Comparison failed, transaction received by the DUT:%s while the refrence out:0b%0b",seq\_item\_sb.convert2string(),seq\_item\_sb.MISO\_G));

                    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

        function void report\_phase(uvm\_phase phase);

            super.report\_phase(phase);

            `uvm\_info("report\_phase",$sformatf("Total successful transaction: %0d",correct\_count),UVM\_MEDIUM);

            `uvm\_info("report\_phase",$sformatf("Total failed transaction: %0d",error\_count),UVM\_MEDIUM);

        endfunction

    endclass

endpackage

monitor

package SPI\_monitor\_pkg;

    import uvm\_pkg::\*;

    import SPI\_seq\_item\_pkg::\*;

    `include "uvm\_macros.svh"

    class SPI\_monitor extends  uvm\_monitor;

        `uvm\_component\_utils(SPI\_monitor)

        virtual SPI\_if SPI\_monitor\_vif;

        SPI\_seq\_item rsp\_seq\_item;

        uvm\_analysis\_port #(SPI\_seq\_item) mon\_ap;

        function new(string name = "SPI\_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 = SPI\_seq\_item::type\_id::create("rsp\_seq\_item");

                @(negedge SPI\_monitor\_vif.clk);

                rsp\_seq\_item.MOSI = SPI\_monitor\_vif.MOSI;

                rsp\_seq\_item.SS\_n = SPI\_monitor\_vif.SS\_n;

                rsp\_seq\_item.rst\_n = SPI\_monitor\_vif.rst\_n;

                rsp\_seq\_item.MISO = SPI\_monitor\_vif.MISO;

                mon\_ap.write(rsp\_seq\_item);

                `uvm\_info("run\_phase",rsp\_seq\_item.convert2string(),UVM\_HIGH);

            end

        endtask

    endclass

endpackage

coverage collector

package SPI\_coverage\_pkg;

    import uvm\_pkg::\*;

    import SPI\_seq\_item\_pkg::\*;

    `include "uvm\_macros.svh"

    class SPI\_coverage extends  uvm\_component;

        `uvm\_component\_utils(SPI\_coverage)

        uvm\_analysis\_export #(SPI\_seq\_item) cov\_export;

        uvm\_tlm\_analysis\_fifo #(SPI\_seq\_item) cov\_fifo;

        SPI\_seq\_item seq\_item\_cov;

        covergroup cg();

            MISO\_coverage:coverpoint seq\_item\_cov.MISO{

            bins ZERO = {0};

            bins ONE  = {1};

            bins WRITE\_adderss = (0 => 0 => 0);

            bins WRITE\_data = (0 => 0 => 1);

            bins READ\_adderss = (1 => 1 => 0);

            bins READ\_data = (1 => 1 => 1);

            }

            SS\_n\_coverage:coverpoint seq\_item\_cov.SS\_n{

            bins ZERO = {0};

            bins ONE  = {1};

            bins ONE\_to\_ZERO = (1 => 0);

            bins ZERO\_to\_ONE  = (0 => 1);

            }

            rst\_n\_coverage:coverpoint seq\_item\_cov.rst\_n{

            bins ZERO = {0};

            bins ONE  = {1};

            }

            MOSI\_coverage:coverpoint seq\_item\_cov.MOSI{

            bins ZERO = {0};

            bins ONE  = {1};

            }

            cross\_1:cross SS\_n\_coverage,rst\_n\_coverage {

            bins bin1 = binsof(SS\_n\_coverage.ONE) && binsof(rst\_n\_coverage.ZERO);

            bins bin2 = binsof(SS\_n\_coverage.ONE\_to\_ZERO) && binsof(rst\_n\_coverage.ZERO);

            bins bin3 = binsof(SS\_n\_coverage.ONE\_to\_ZERO) && binsof(rst\_n\_coverage.ONE);

            option.cross\_auto\_bin\_max = 0;

            }

            cross\_2:cross MISO\_coverage,SS\_n\_coverage{

            bins bin1 = binsof(MISO\_coverage.WRITE\_adderss) && binsof(SS\_n\_coverage.ZERO);

            bins bin2 = binsof(MISO\_coverage.WRITE\_data) && binsof(SS\_n\_coverage.ZERO);

            bins bin3 = binsof(MISO\_coverage.READ\_adderss) && binsof(SS\_n\_coverage.ZERO);

            bins bin4 = binsof(MISO\_coverage.READ\_data) && binsof(SS\_n\_coverage.ZERO);

            option.cross\_auto\_bin\_max = 0;

            }

        endgroup

        function new(string name = "SPI\_coverage",uvm\_component parent = null);

            super.new(name,parent);

            cg = new();

        endfunction

        function void build\_phase(uvm\_phase phase);

            super.build\_phase(phase);

            cov\_export = new("cov\_export",this);

            cov\_fifo = new("cov\_fifo",this);

        endfunction

        function void connect\_phase(uvm\_phase phase);

            super.connect\_phase(phase);

            cov\_export.connect(cov\_fifo.analysis\_export);

        endfunction

        task run\_phase(uvm\_phase phase);

            super.run\_phase(phase);

            forever begin

                cov\_fifo.get(seq\_item\_cov);

                cg.sample();

            end

        endtask

    endclass

endpackage