Final UVM Project

Part (2)-SPI-Wrapper-Environment:

1. Verification plan

a) Reset functionality should clear the flag, counter, rx_valid and MISO.
b) Check Starting communication phase when SS_n gets low.
c) Testing next state choice after current state is check command according to flag & MOSI.
d) Verify the serial to parallel conversion operation after 10 clock cycles after its start.
e) Checking the address is updated after the rx_valid is asserted in WRITE and READ_ADD cases.
f) Verifying the Read_data process.
g) Check End communication transition when master makes SS_n high.

Note:

- In Constraining and debugging the possibility of the master delaying the end communication signal (SS_n) was not taken in consideration for example if the master is writing in the RAM the high SS_n signal will come right after 11 clock cycles from starting the communication.

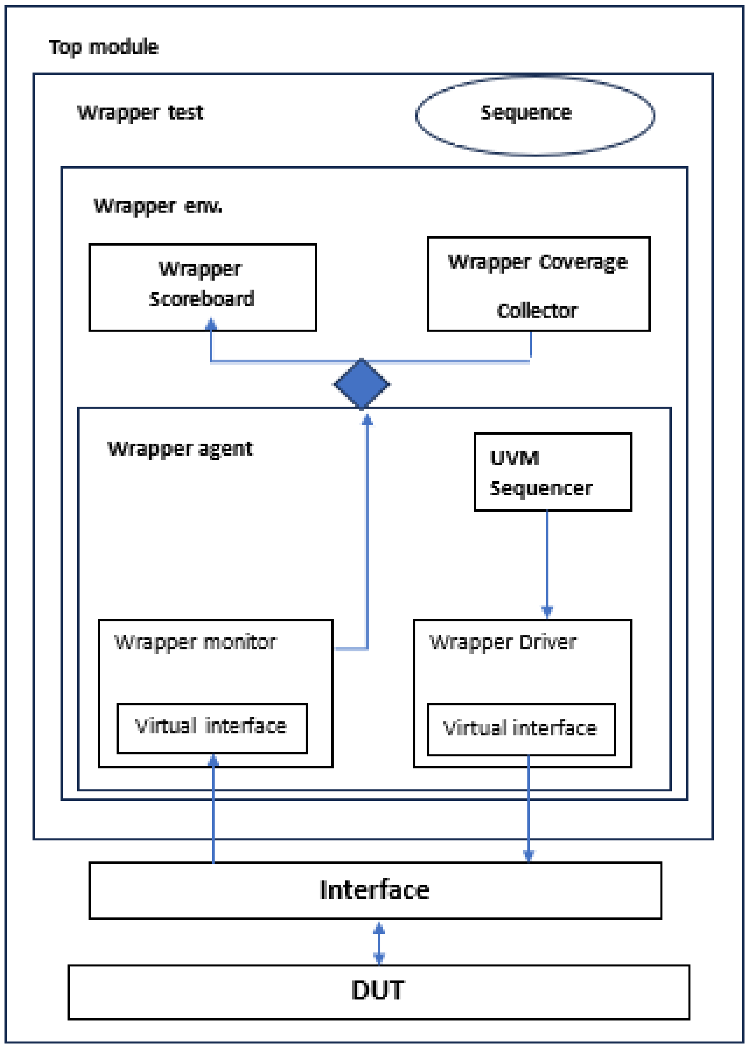
- In this DUT: dividing the inter tasks (counter, PISO, SIPO) in the SPI module resulted in inevitable delays that are not specified in the specs to correct these delays we would have to tear down the whole design. So, we just test for basic functionality neglecting timing specially in READ_DATA case as it clearly has a problem.

- We adjusted the testbench to match the design delays in order to check for the delayed output each clock cycle.

➢ <u>UVM Structure:</u>

Note:

- In stimulus generation the word (in 11 or 18 clock cycles) sent to the wrapper by the master is randomized only once then in the driver class the MOSI is assigned from this word one by one each negative edge of the clock.
- In Scoreboard, checking output takes place only in READ_DATA case where the MISO is collected in a variable called word_rec then it is compared at the end with the word expected.

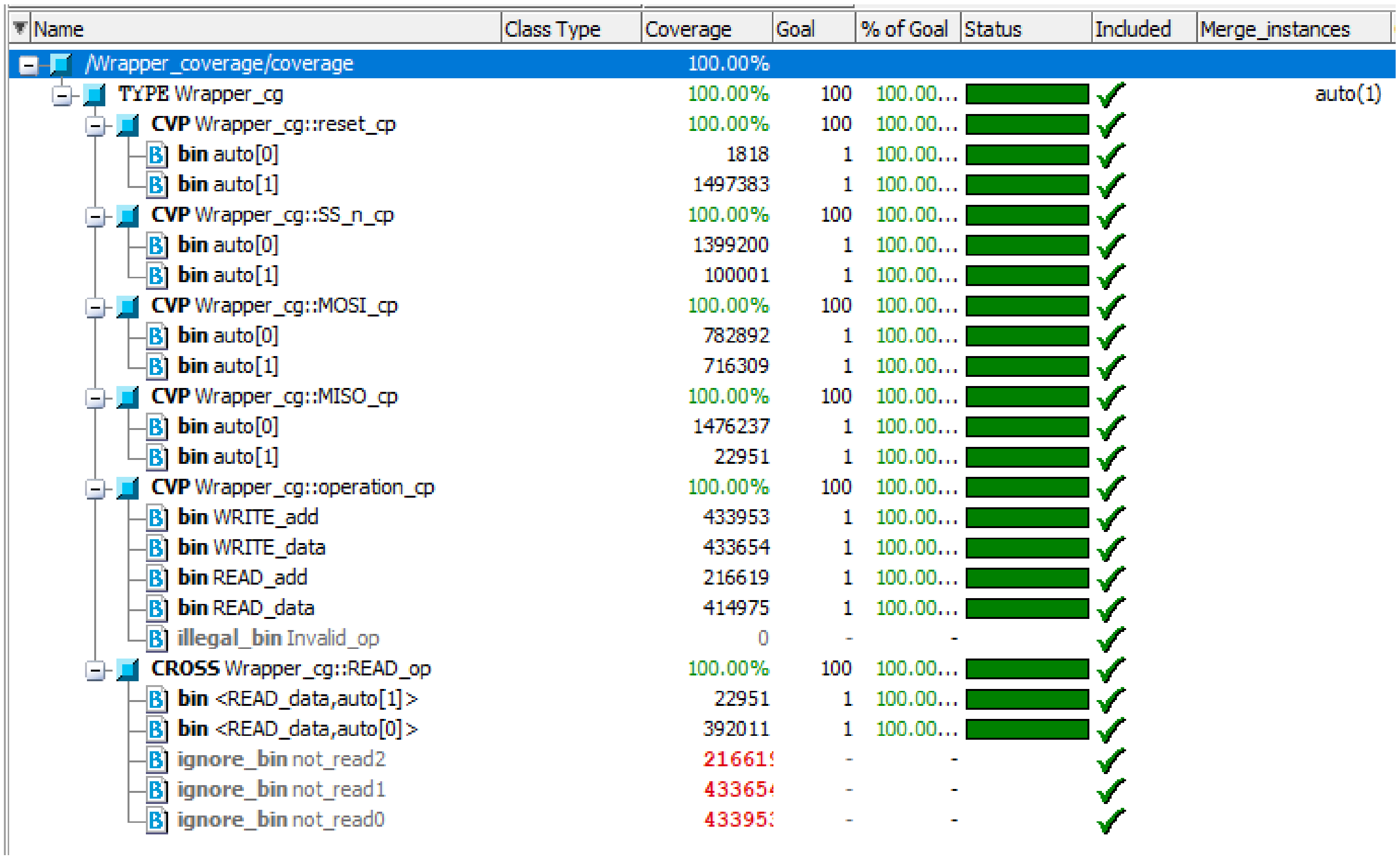


UVM Structure diagram

2. Coverage reports:

```
Coverage Report by instance with details

=================================================================================
=== Instance: /\top#DUT /spislave/shift_reg
=== Design Unit: work.SIPO
=================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         2         2         0   100.00%

================================Branch Details================================
    Statement Coverage:
        Enabled Coverage              Bins      Hits    Misses  Coverage
        ----------------              ----      ----    ------  --------
        Statements                       2         2         0   100.00%

    ================================Statement Details================================


    Toggle Coverage:
        Enabled Coverage              Bins      Hits    Misses  Coverage
        ----------------              ----      ----    ------  --------
        Toggles                         46        46         0   100.00%

    ================================Toggle Details================================


    FSM Coverage:
        Enabled Coverage              Bins      Hits    Misses  Coverage
        ----------------              ----      ----    ------  --------
        FSM States                       5         5         0   100.00%
        FSM Transitions                  8         7         1    87.50%

    ================================FSM Details================================


    Toggle Coverage:
        Enabled Coverage              Bins      Hits    Misses  Coverage
        ----------------              ----      ----    ------  --------
        Toggles                         96        96         0   100.00%

    ================================Toggle Details================================

  =================================================================================
  === Instance: /\top#DUT /mem
  === Design Unit: work.project_ram
  =================================================================================
  Branch Coverage:
      Enabled Coverage              Bins      Hits    Misses  Coverage
      ----------------              ----      ----    ------  --------
      Branches                         7         7         0   100.00%

  ================================Branch Details================================

 Statement Coverage:
     Enabled Coverage              Bins      Hits    Misses  Coverage
     ----------------              ----      ----    ------  --------
     Statements                      19        19         0   100.00%

 ================================Statement Details================================
```

➢ Function Coverage:



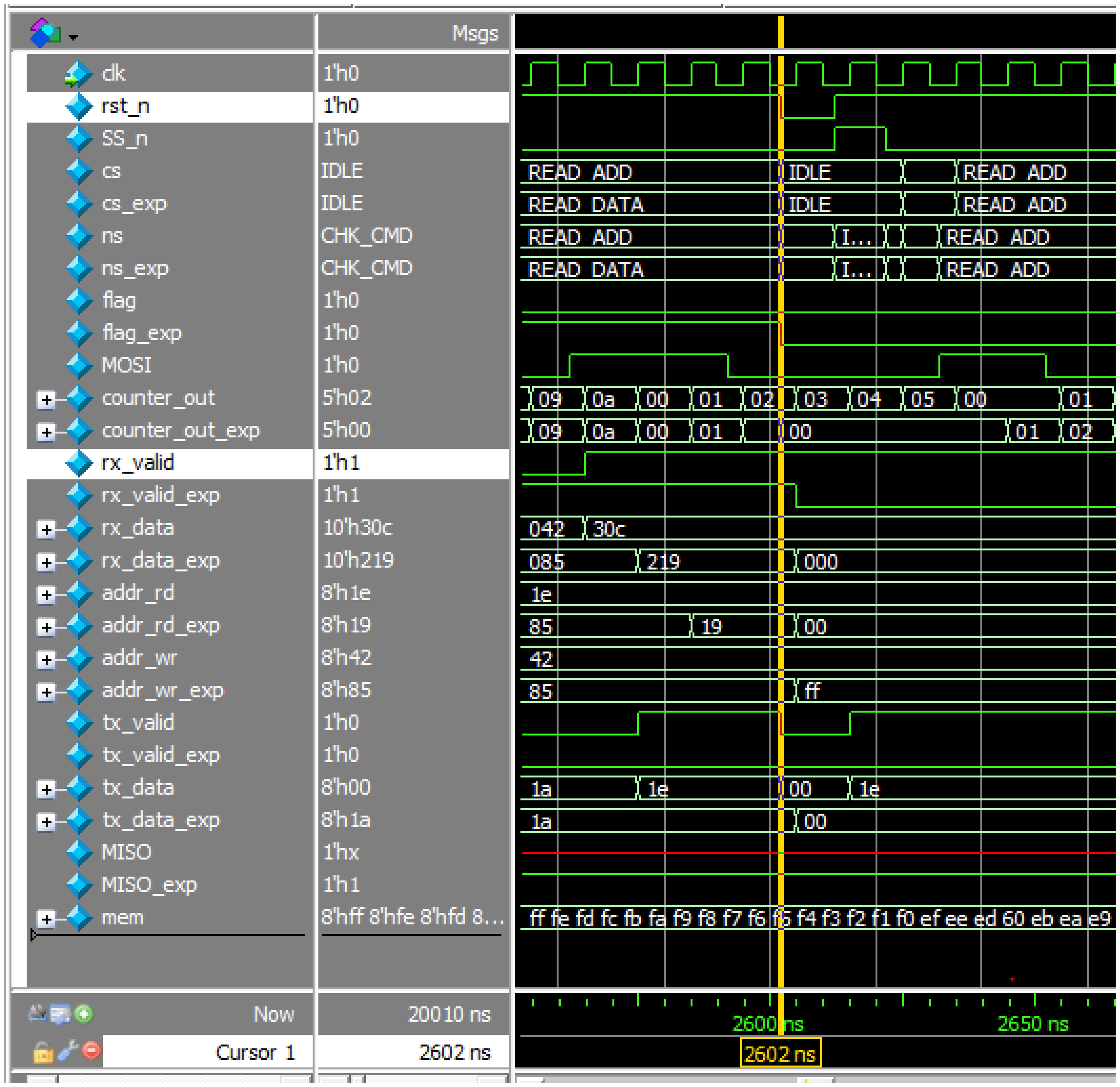
Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances
/Wrapper_coverage/coverage		100.00%					
TYPE Wrapper_cg		100.00%	100	100.00...	🟩	✓	auto(1)
CVP Wrapper_cg::reset_cp		100.00%	100	100.00...	🟩	✓	
bin auto[0]		1818	1	100.00...	🟩	✓	
bin auto[1]		1497383	1	100.00...	🟩	✓	
CVP Wrapper_cg::SS_n_cp		100.00%	100	100.00...	🟩	✓	
bin auto[0]		1399200	1	100.00...	🟩	✓	
bin auto[1]		100001	1	100.00...	🟩	✓	
CVP Wrapper_cg::MOSI_cp		100.00%	100	100.00...	🟩	✓	
bin auto[0]		782892	1	100.00...	🟩	✓	
bin auto[1]		716309	1	100.00...	🟩	✓	
CVP Wrapper_cg::MISO_cp		100.00%	100	100.00...	🟩	✓	
bin auto[0]		1476237	1	100.00...	🟩	✓	
bin auto[1]		22951	1	100.00...	🟩	✓	
CVP Wrapper_cg::operation_cp		100.00%	100	100.00...	🟩	✓	
bin WRITE_add		433953	1	100.00...	🟩	✓	
bin WRITE_data		433654	1	100.00...	🟩	✓	
bin READ_add		216619	1	100.00...	🟩	✓	
bin READ_data		414975	1	100.00...	🟩	✓	
illegal_bin Invalid_op		0	-	-		✓	
CROSS Wrapper_cg::READ_op		100.00%	100	100.00...	🟩	✓	
bin <READ_data,auto[1]>		22951	1	100.00...	🟩	✓	
bin <READ_data,auto[0]>		392011	1	100.00...	🟩	✓	
ignore_bin not_read2		21661	-	-		✓	
ignore_bin not_read1		43365	-	-		✓	
ignore_bin not_read0		43395	-	-		✓	

Note: Coverage didn’t reach 100% because of the assumption mentioned above that the master does not end the communication early and because of the all-false condition.
Also, there is no assertions for this part.

3. Bugs report:

a) Timing delays in the SPI wrapper operations specially in READ_DATA case.

b) In case of active reset rx_valid does not change (remains the same).

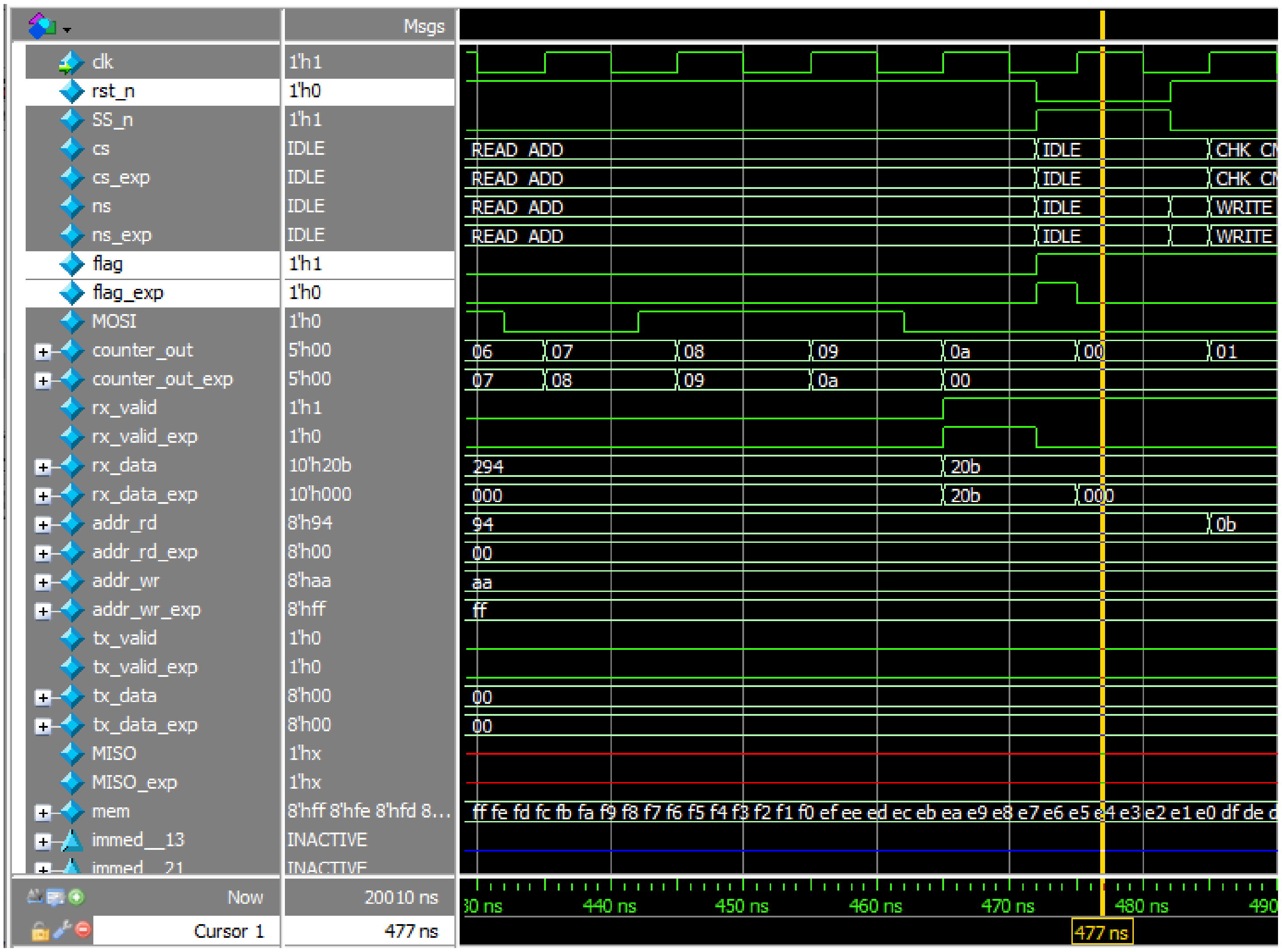


	Msgs
clk	1'h0
rst_n	1'h0
SS_n	1'h0
cs	IDLE
cs_exp	IDLE
ns	CHK_CMD
ns_exp	CHK_CMD
flag	1'h0
flag_exp	1'h0
MOSI	1'h0
counter_out	5'h02
counter_out_exp	5'h00
rx_valid	1'h1
rx_valid_exp	1'h1
rx_data	10'h30c
rx_data_exp	10'h219
addr_rd	8'h1e
addr_rd_exp	8'h19
addr_wr	8'h42
addr_wr_exp	8'h85
tx_valid	1'h0
tx_valid_exp	1'h0
tx_data	8'h00
tx_data_exp	8'h1a
MISO	1'hx
MISO_exp	1'h1
mem	8'hff 8'hfe 8'hfd 8...

Now	20010 ns	
Cursor 1	2602 ns	2602 ns

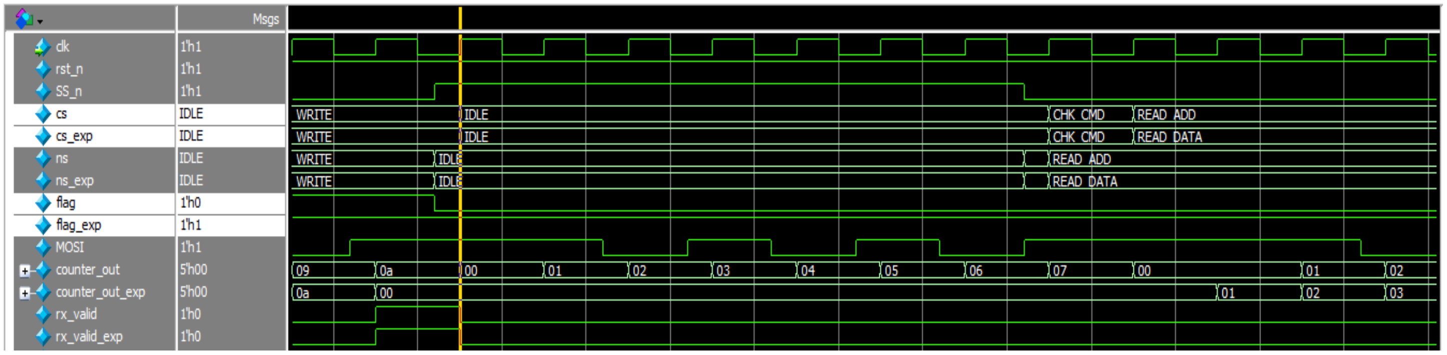
Snippet shows the stability of the rx_valid flag even during active reset

c) During asserted reset the flag which indicates that an address was sent does not get cleared.



Snippet shows the stability of the address flag even during active reset

d) If the current state is WRITE and master ends the communication the flag is cleared. Which causes a problem in the next READ operation induces wrong READ_ADD operation.

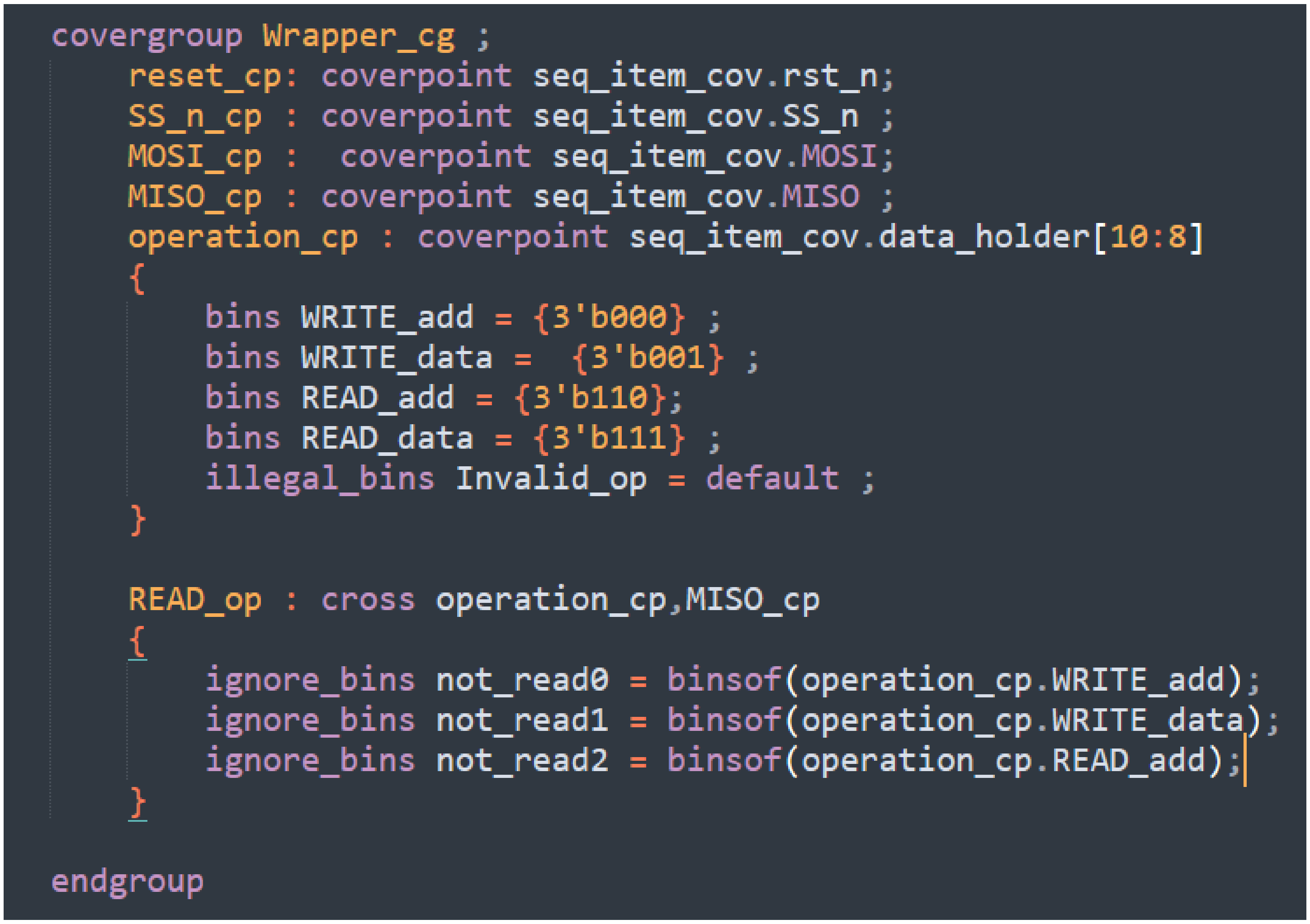


Snippet shows the false clear of the flag after WRITE operation

e) Checking to clear the counter in the design takes place on the current state which makes the counter clears but the next clock cycle instead it should check for the next state if it is equal CHK_CMD to clear the counter right away and be able to assert rx_valid and access memory before the communication ends.

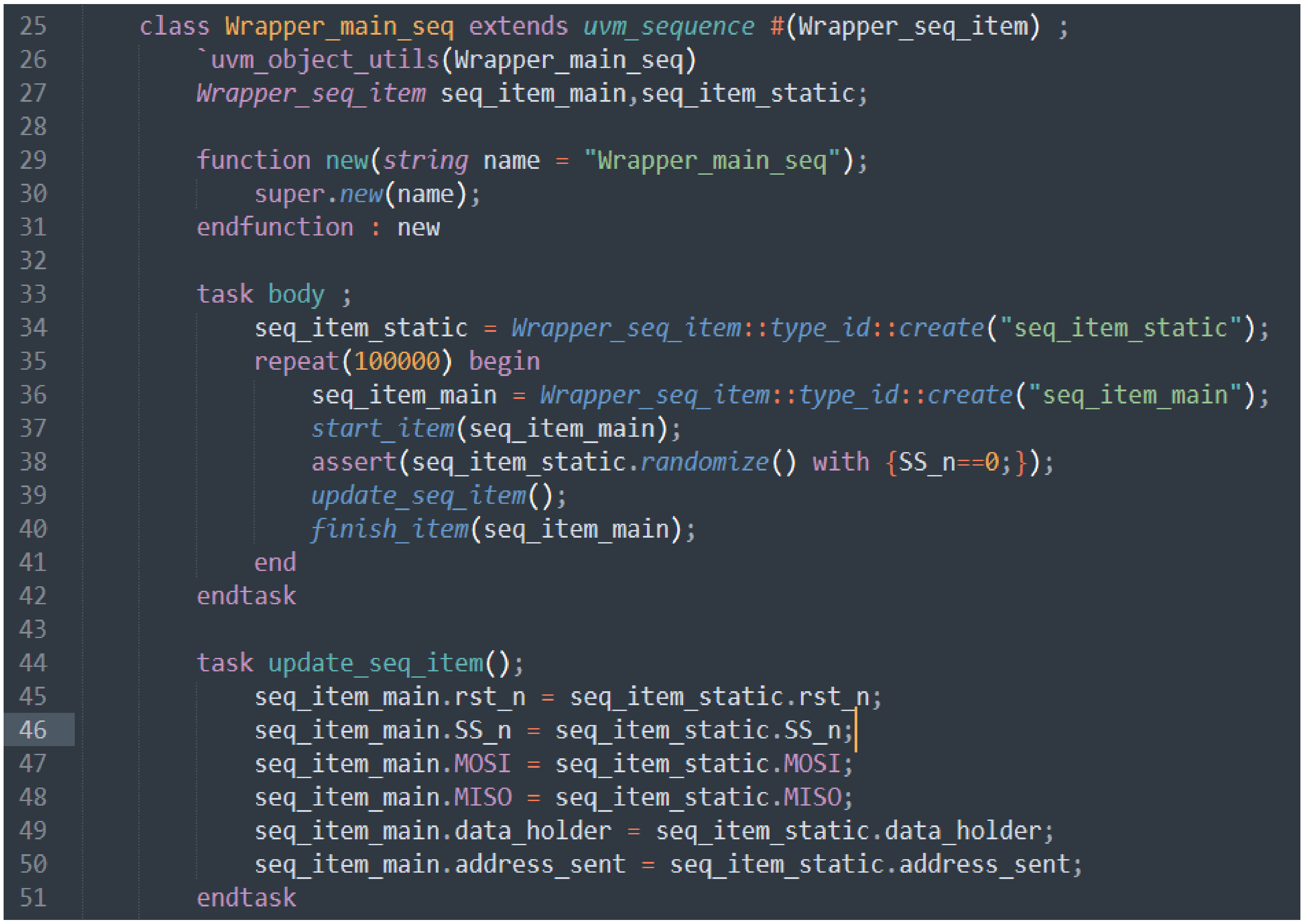
4. Code snippets

a) Coverage & cover points:



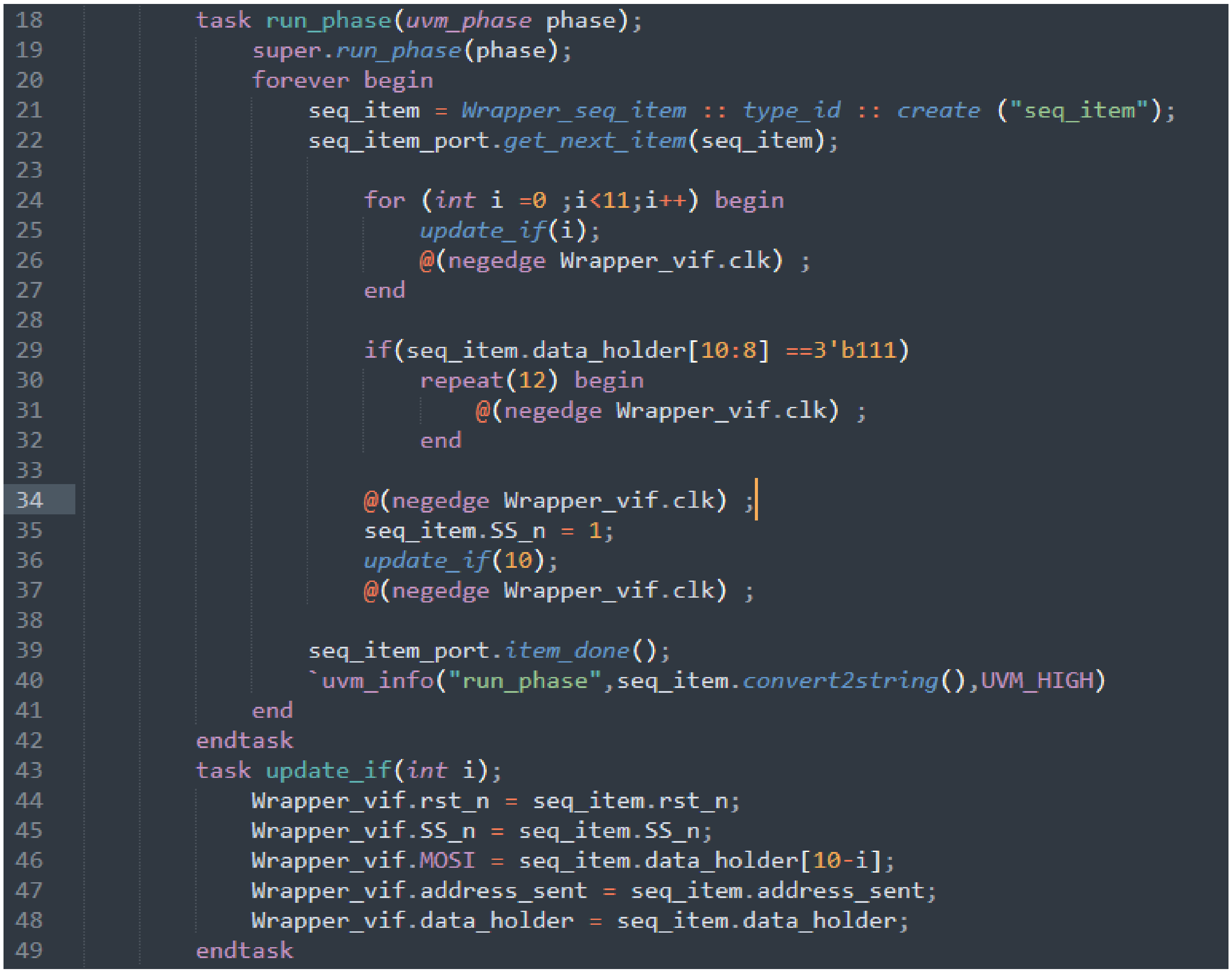
```
covergroup Wrapper_cg ;
    reset_cp: coverpoint seq_item_cov.rst_n;
    SS_n_cp : coverpoint seq_item_cov.SS_n ;
    MOSI_cp :  coverpoint seq_item_cov.MOSI;
    MISO_cp : coverpoint seq_item_cov.MISO ;
    operation_cp : coverpoint seq_item_cov.data_holder[10:8]
    {
        bins WRITE_add = {3'b000} ;
        bins WRITE_data =  {3'b001} ;
        bins READ_add = {3'b110};
        bins READ_data = {3'b111} ;
        illegal_bins Invalid_op = default ;
    }

    READ_op : cross operation_cp,MISO_cp
    {
        ignore_bins not_read0 = binsof(operation_cp.WRITE_add);
        ignore_bins not_read1 = binsof(operation_cp.WRITE_data);
        ignore_bins not_read2 = binsof(operation_cp.READ_add);
    }

endgroup
```

b) Stimulus driving:



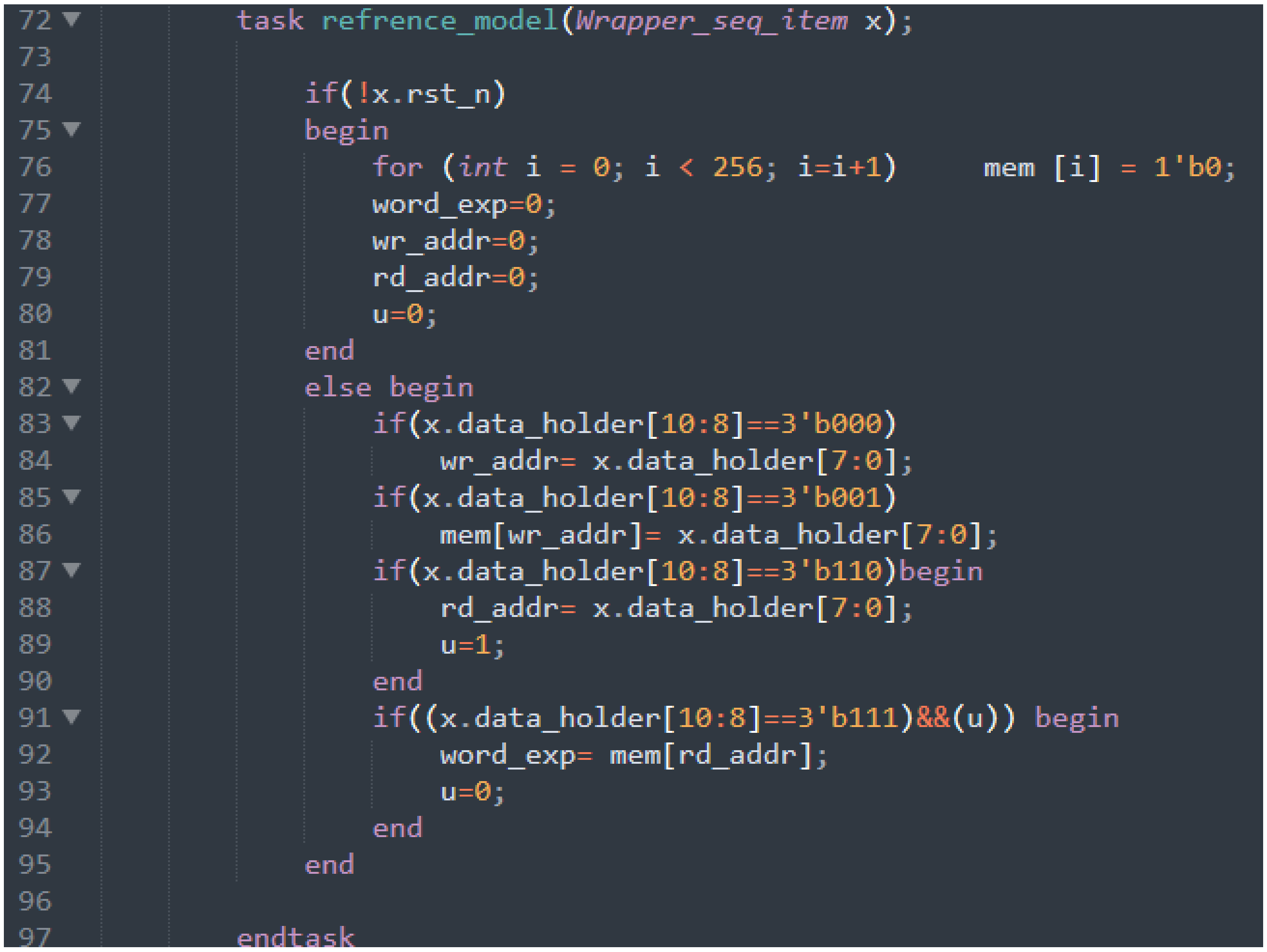
```systemverilog
class Wrapper_main_seq extends uvm_sequence #(Wrapper_seq_item) ;
    `uvm_object_utils(Wrapper_main_seq)
    Wrapper_seq_item seq_item_main,seq_item_static;

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

    task body ;
        seq_item_static = Wrapper_seq_item::type_id::create("seq_item_static");
        repeat(100000) begin
            seq_item_main = Wrapper_seq_item::type_id::create("seq_item_main");
            start_item(seq_item_main);
            assert(seq_item_static.randomize() with {SS_n==0;});
            update_seq_item();
            finish_item(seq_item_main);
        end
    endtask

    task update_seq_item();
        seq_item_main.rst_n = seq_item_static.rst_n;
        seq_item_main.SS_n = seq_item_static.SS_n;
        seq_item_main.MOSI = seq_item_static.MOSI;
        seq_item_main.MISO = seq_item_static.MISO;
        seq_item_main.data_holder = seq_item_static.data_holder;
        seq_item_main.address_sent = seq_item_static.address_sent;
    endtask
```

Snippet shows the sequence class


```
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                seq_item = Wrapper_seq_item :: type_id :: create ("seq_item");
                seq_item_port.get_next_item(seq_item);

                    for (int i =0 ;i<11;i++) begin
                        update_if(i);
                        @(negedge Wrapper_vif.clk) ;
                    end

                    if(seq_item.data_holder[10:8] ==3'b111)
                        repeat(12) begin
                            @(negedge Wrapper_vif.clk) ;
                        end

                    @(negedge Wrapper_vif.clk) ;
                    seq_item.SS_n = 1;
                    update_if(10);
                    @(negedge Wrapper_vif.clk) ;

                seq_item_port.item_done();
                `uvm_info("run_phase",seq_item.convert2string(),UVM_HIGH)
            end
        endtask
        task update_if(int i);
            Wrapper_vif.rst_n = seq_item.rst_n;
            Wrapper_vif.SS_n = seq_item.SS_n;
            Wrapper_vif.MOSI = seq_item.data_holder[10-i];
            Wrapper_vif.address_sent = seq_item.address_sent;
            Wrapper_vif.data_holder = seq_item.data_holder;
        endtask
```

Snippet shows the wrapper driver

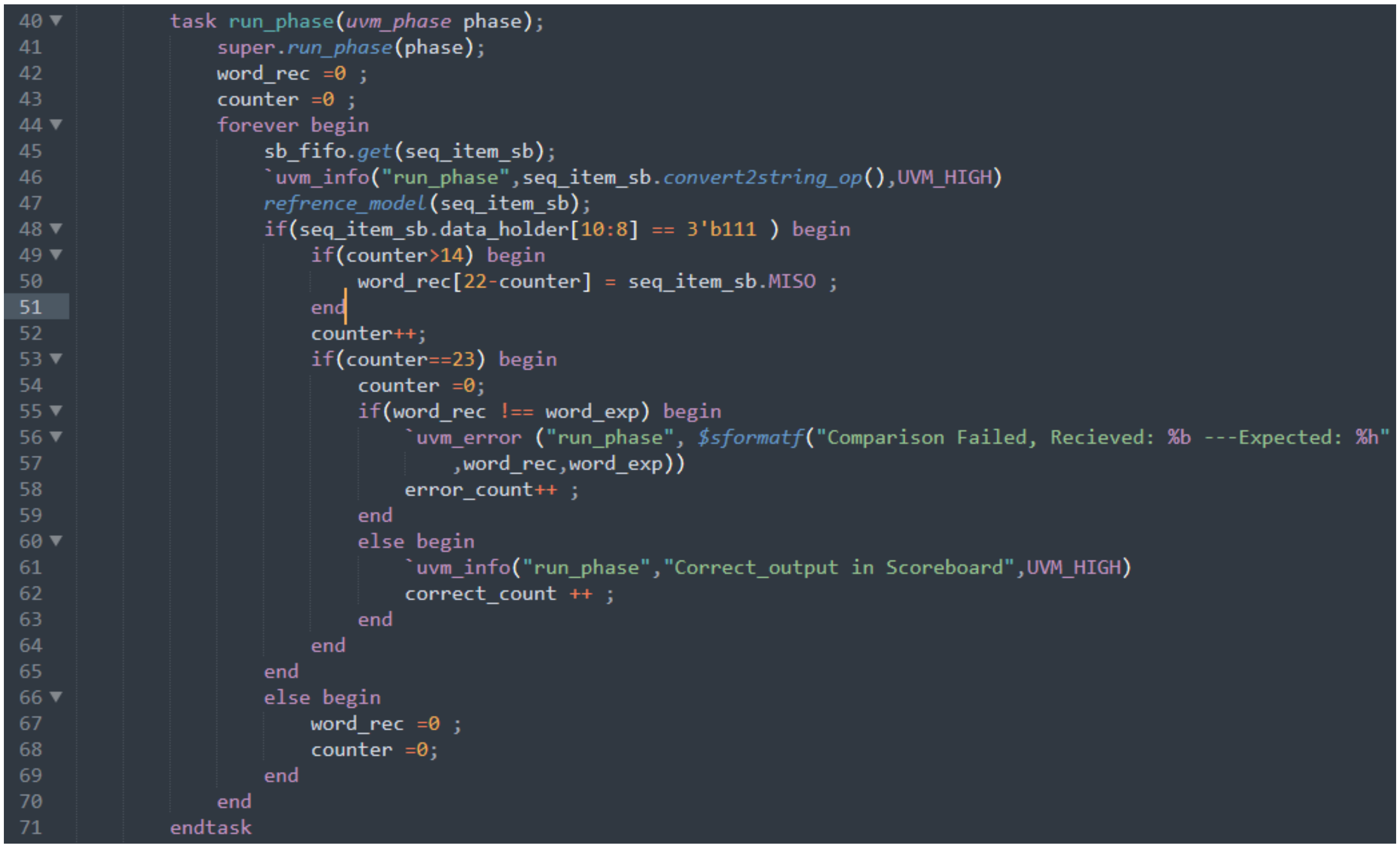
c) Reference model:



```
        task refrence_model(Wrapper_seq_item x);

            if(!x.rst_n)
            begin
                for (int i = 0; i < 256; i=i+1)     mem [i] = 1'b0;
                word_exp=0;
                wr_addr=0;
                rd_addr=0;
                u=0;
            end
            else begin
                if(x.data_holder[10:8]==3'b000)
                    wr_addr= x.data_holder[7:0];
                if(x.data_holder[10:8]==3'b001)
                    mem[wr_addr]= x.data_holder[7:0];
                if(x.data_holder[10:8]==3'b110)begin
                    rd_addr= x.data_holder[7:0];
                    u=1;
                end
                if((x.data_holder[10:8]==3'b111)&&(u)) begin
                    word_exp= mem[rd_addr];
                    u=0;
                end
            end

        endtask
```

Snippet shows the reference model task that updates the expected values

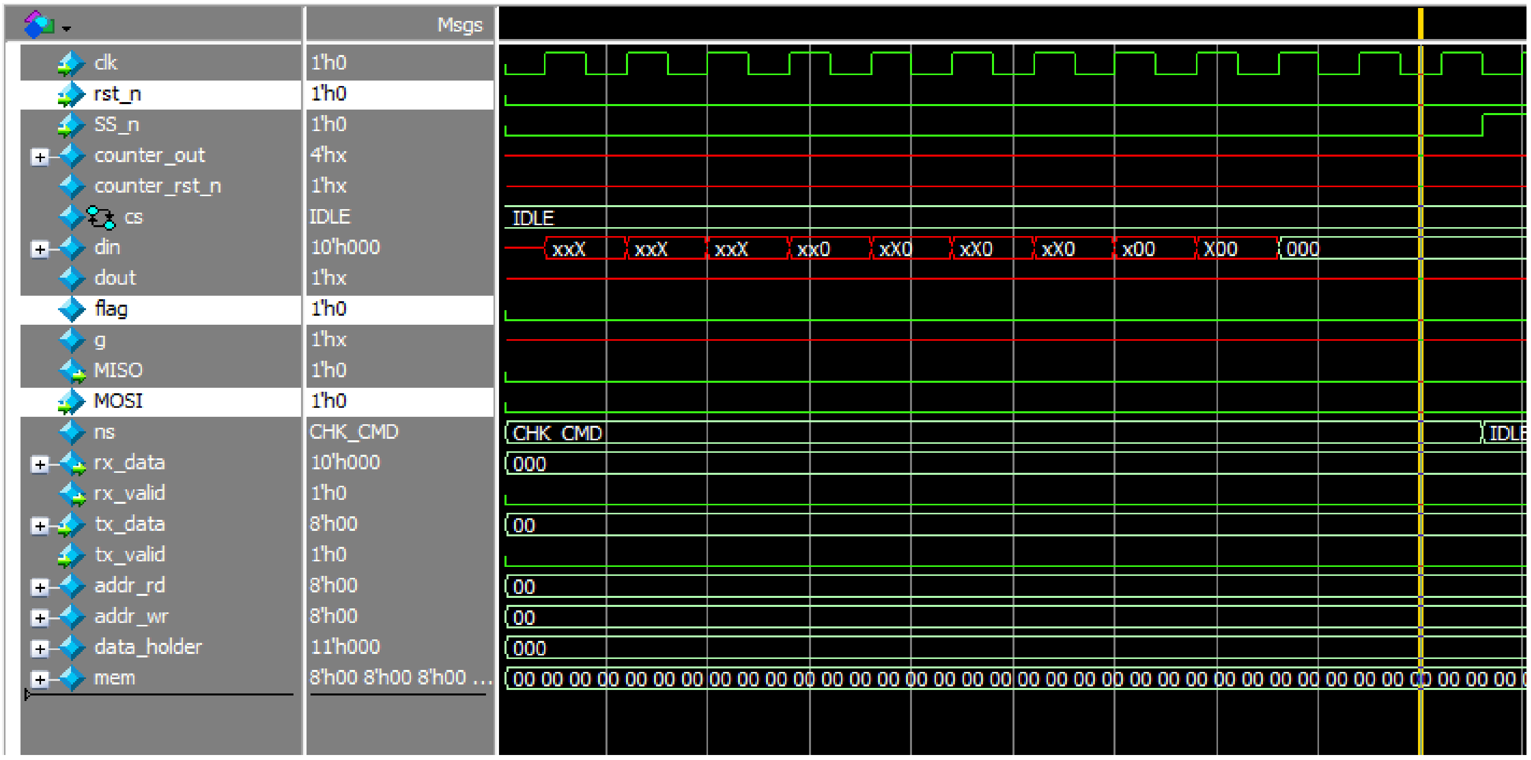
d) Output checking:



```
    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        word_rec =0 ;
        counter =0 ;
        forever begin
            sb_fifo.get(seq_item_sb);
            `uvm_info("run_phase",seq_item_sb.convert2string_op(),UVM_HIGH)
            refrence_model(seq_item_sb);
            if(seq_item_sb.data_holder[10:8] == 3'b111 ) begin
                if(counter>14) begin
                    word_rec[22-counter] = seq_item_sb.MISO ;
                end
                counter++;
                if(counter==23) begin
                    counter =0;
                    if(word_rec !== word_exp) begin
                        `uvm_error ("run_phase", $sformatf("Comparison Failed, Recieved: %b ---Expected: %h"
                            ,word_rec,word_exp))
                        error_count++ ;
                    end
                    else begin
                        `uvm_info("run_phase","Correct_output in Scoreboard",UVM_HIGH)
                        correct_count ++ ;
                    end
                end
            end
            else begin
                word_rec =0 ;
                counter =0;
            end
        end
    endtask
```

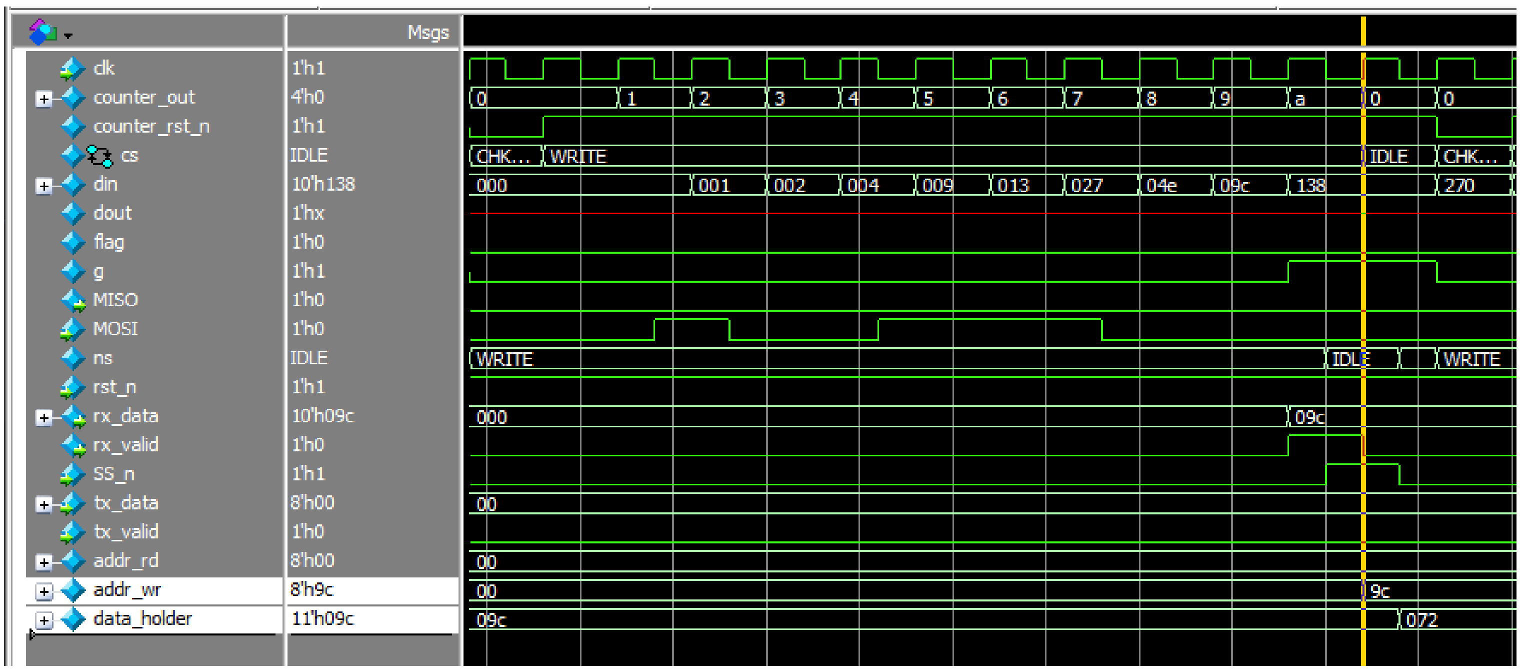
Snippet shows the comparing of the output data in scoreboard

5. Questa snippets:



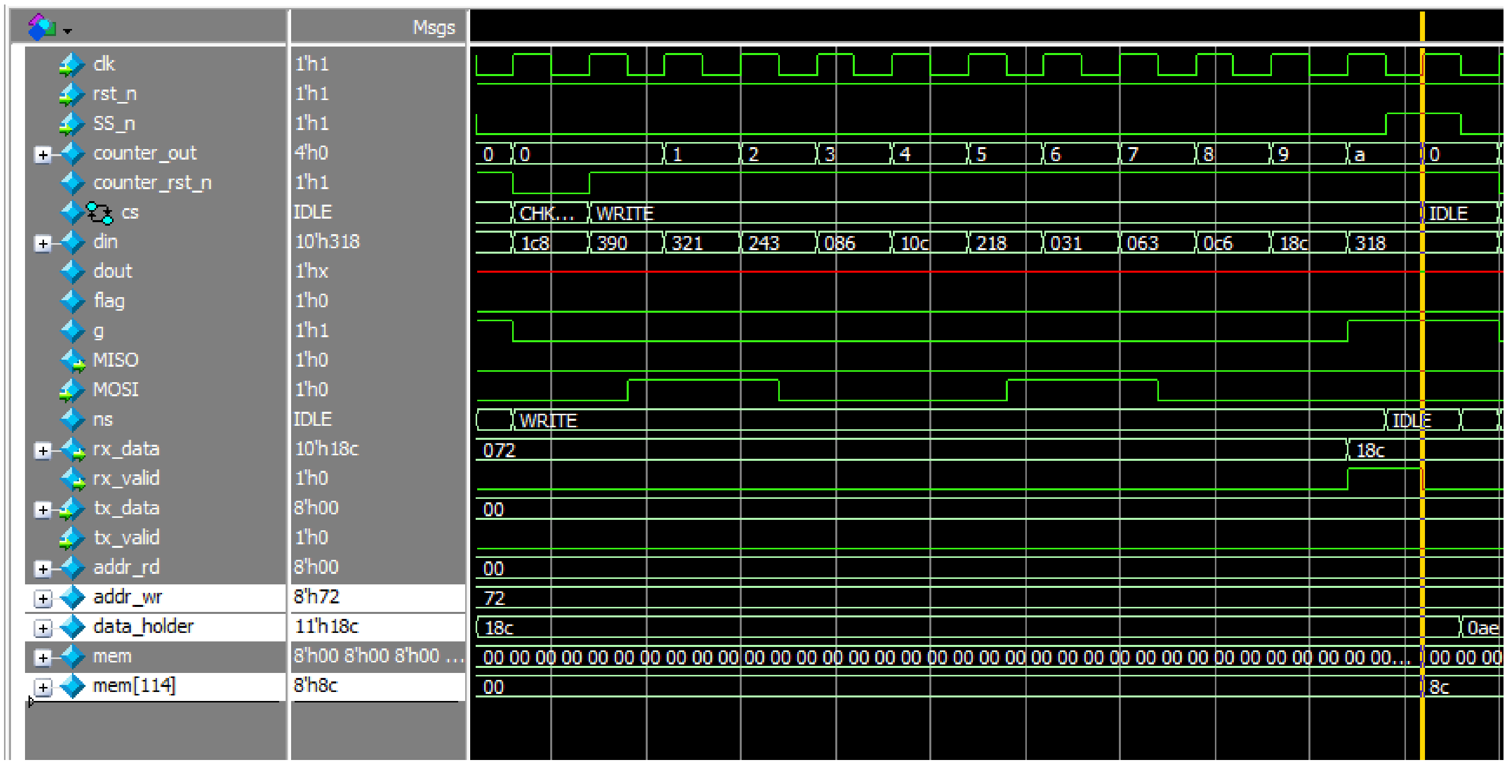
```
                Msgs
clk             1'h0
rst_n           1'h0
SS_n            1'h0
counter_out     4'hx
counter_rst_n   1'hx
cs              IDLE
din             10'h000
dout            1'hx
flag            1'h0
g               1'hx
MISO            1'h0
MOSI            1'h0
ns              CHK_CMD
rx_data         10'h000
rx_valid        1'h0
tx_data         8'h00
tx_valid        1'h0
addr_rd         8'h00
addr_wr         8'h00
data_holder     11'h000
mem             8'h00 8'h00 8'h00 ...
```

Waveform shows the Reset functionality



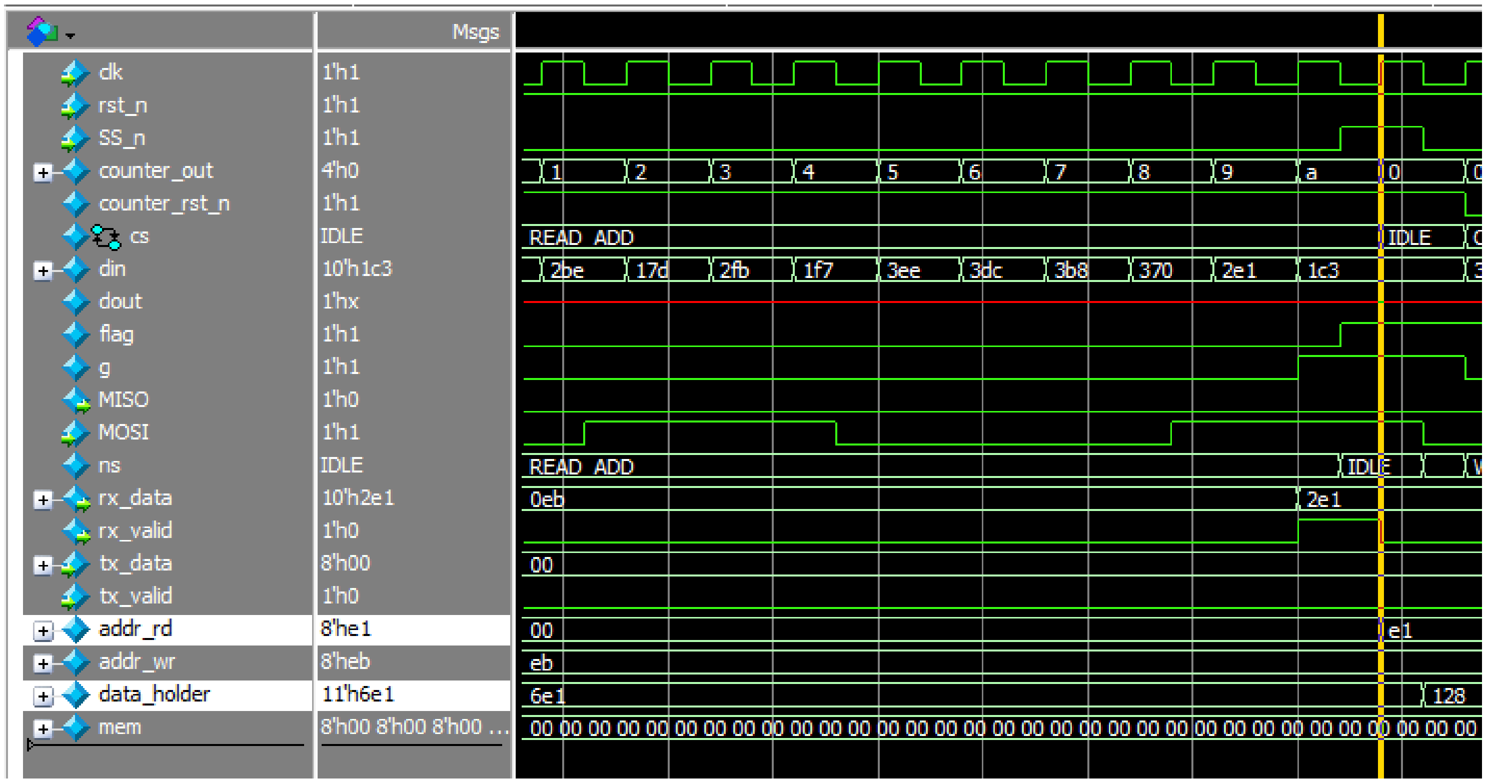
	Msgs
clk	1'h1
counter_out	4'h0
counter_rst_n	1'h1
cs	IDLE
din	10'h138
dout	1'hx
flag	1'h0
g	1'h1
MISO	1'h0
MOSI	1'h0
ns	IDLE
rst_n	1'h1
rx_data	10'h09c
rx_valid	1'h0
SS_n	1'h1
tx_data	8'h00
tx_valid	1'h0
addr_rd	8'h00
addr_wr	8'h9c
data_holder	11'h09c

Waveform shows the write address operation



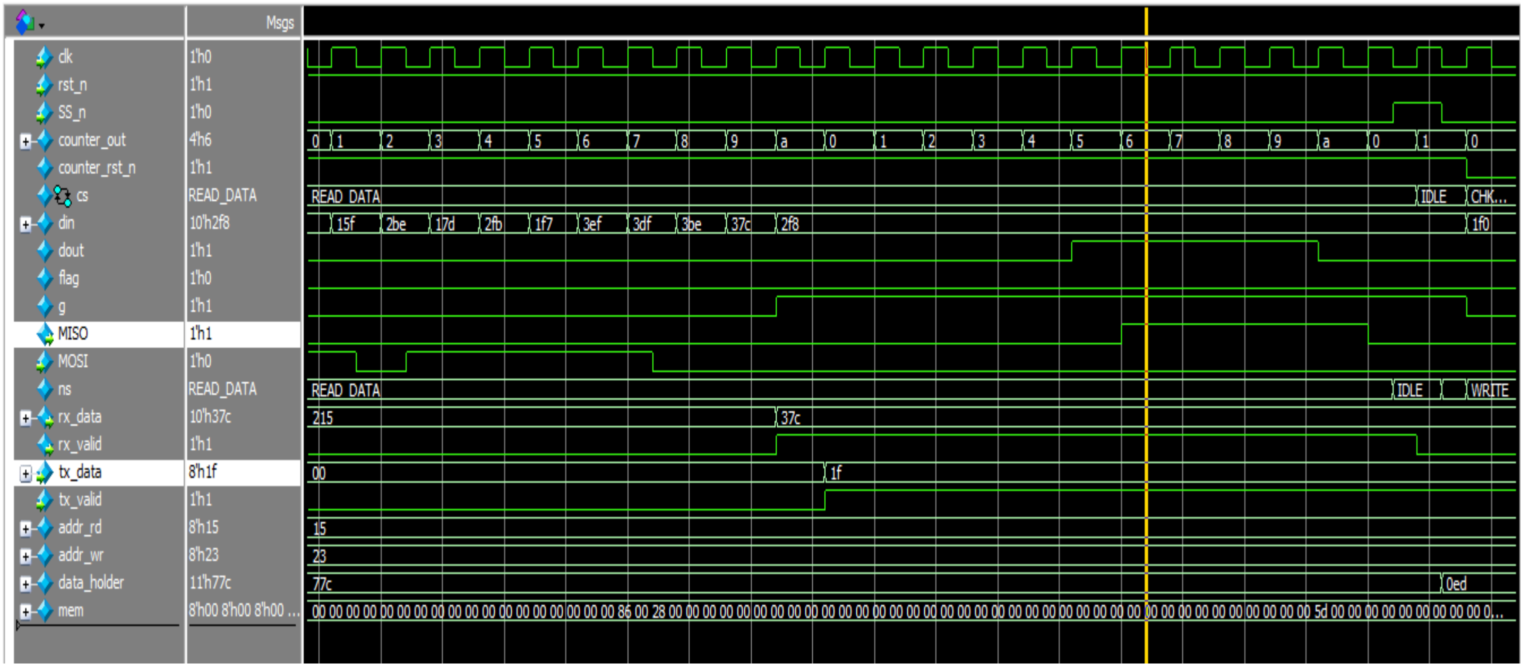
	Msgs
clk	1'h1
rst_n	1'h1
SS_n	1'h1
counter_out	4'h0
counter_rst_n	1'h1
cs	IDLE
din	10'h318
dout	1'hx
flag	1'h0
g	1'h1
MISO	1'h0
MOSI	1'h0
ns	IDLE
rx_data	10'h18c
rx_valid	1'h0
tx_data	8'h00
tx_valid	1'h0
addr_rd	8'h00
addr_wr	8'h72
data_holder	11'h18c
mem	8'h00 8'h00 8'h00 ...
mem[114]	8'h8c

Waveform shows the write data operation



```
                Msgs
clk             1'h1
rst_n           1'h1
SS_n            1'h1
counter_out     4'h0
counter_rst_n   1'h1
cs              IDLE
din             10'h1c3
dout            1'hx
flag            1'h1
g               1'h1
MISO            1'h0
MOSI            1'h1
ns              IDLE
rx_data         10'h2e1
rx_valid        1'h0
tx_data         8'h00
tx_valid        1'h0
addr_rd         8'he1
addr_wr         8'heb
data_holder     11'h6e1
mem             8'h00 8'h00 8'h00 ...
```

Waveform shows the read address operation



```
                Msgs
clk             1'h0
rst_n           1'h1
SS_n            1'h0
counter_out     4'h6
counter_rst_n   1'h1
cs              READ_DATA
din             10'h2f8
dout            1'h1
flag            1'h0
g               1'h1
MISO            1'h1
MOSI            1'h0
ns              READ_DATA
rx_data         10'h37c
rx_valid        1'h1
tx_data         8'h1f
tx_valid        1'h1
addr_rd         8'h15
addr_wr         8'h23
data_holder     11'h77c
mem             8'h00 8'h00 8'h00 ...
```

Waveform shows the read data operation and the delays caused by the submodules (dout & MISO)