Omar Nour Eldin Mohamed Khalil

Synchronous FIFO

SV project

FIFO design before modification has 3 Bugs which are:

Underflow is designed to be combinational
Almostfull is assigned to be (FIFO_depth-2)
Reset doesn't clear all the sequential flags

Verification plan:

Randomize input data and pass randomized values to object from FIFO_transaction class
Then that object will be passed to two parallel threads (using fork join) ➔sample function and check data function
Check data will compare the DUT and reference objects
Reference function will be built
If the data is correct, contents of the queue will be displayed and correct_count ++
Else, an error message will be displayed and error_count++
A test_finished signal is asserted in the Testbench
If test_finished signal == 1, then simulation stops

RTL design with bugs:

FIFO_project > fifo_beforeeeeee.sv

```
////////////////////////////////////////////////////////////////////////////////
// Author: Kareem Waseem
// Course: Digital Verification using SV & UVM
//
// Description: FIFO Design ============================================== with bugs ===============
//
////////////////////////////////////////////////////////////////////////////////
module FIFO(FIFO_if.fifo_dut interface_object);
// parameter interface_object.FIFO_WIDTH = 16;
// parameter interface_object.FIFO_DEPTH = 8;
// input [interface_object.FIFO_WIDTH-1:0] interface_object.data_in;
// input interface_object.clk, interface_object.rst_n, interface_object.wr_en, interface_object.rd_en;
// output reg [interface_object.FIFO_WIDTH-1:0] interface_object.data_out;
// output reg interface_object.wr_ack, interface_object.overflow;
// output interface_object.full, interface_object.empty, almostinterface_object.full, almostempty, underflow;

localparam max_fifo_addr = $clog2(interface_object.FIFO_DEPTH);

reg [interface_object.FIFO_WIDTH-1:0] mem [interface_object.FIFO_DEPTH-1:0];

reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [max_fifo_addr:0] count;
```

```
always @(posedge interface_object.clk or negedge interface_object.rst_n) begin
    if (!interface_object.rst_n) begin
        wr_ptr <= 0;
    end
    else if (interface_object.wr_en && count < interface_object.FIFO_DEPTH) begin
        mem[wr_ptr] <= interface_object.data_in;
        interface_object.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    end
    else begin
        interface_object.wr_ack <= 0;
        if (interface_object.full & interface_object.wr_en)
            interface_object.overflow <= 1;
        else
            interface_object.overflow <= 0;
    end
end

```

```
always @(posedge interface_object.clk or negedge interface_object.rst_n) begin
    if (!interface_object.rst_n) begin
        rd_ptr <= 0;
    end
    else if (interface_object.rd_en && count != 0) begin
        interface_object.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
end

always @(posedge interface_object.clk or negedge interface_object.rst_n) begin
    if (!interface_object.rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({interface_object.wr_en, interface_object.rd_en} == 2'b10) && !interface_object.full)
            count <= count + 1;
        else if ( ({interface_object.wr_en, interface_object.rd_en} == 2'b01) && !interface_object.empty)
            count <= count - 1;
    end
end
```

```

  assign interface_object.full = (count == interface_object.FIFO_DEPTH)? 1 : 0;
  assign interface_object.empty = (count == 0)? 1 : 0;
  assign underflow = (interface_object.empty && interface_object.rd_en)? 1 : 0;
  assign interface_object.almostfull = (count == interface_object.FIFO_DEPTH-2)? 1 : 0;
  assign interface_object.almostempty = (count == 1)? 1 : 0;
```

```
property p1;
    @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (interface_object.wr_en && !interface_object.full) |=> interface_object.wr_ack ;
    endproperty
    write_acknowledge : assert property(p1);
    write_acknowledge_cover : cover property(p1);

    property p2;
    @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (interface_object.empty && interface_object.rd_en) |=> interface_object.underflow ;
    endproperty
    underflow_assertion : assert property(p2);
    underflow_cover : cover property(p2);

    property p3;
    @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (interface_object.full & interface_object.wr_en) |=> interface_object.overflow ;
    endproperty
    overflow_assertion : assert property(p3);
    overflow_cover : cover property(p3);

    property p4;
    @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (!interface_object.full & interface_object.wr_en & !interface_object.rd_en) |=> (count == $past(count) + 4'b0001 ) ;
    endproperty
    increment_assertion : assert property(p4);
    increment_cover : cover property(p4);

    property p5;
    @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (!interface_object.empty && interface_object.rd_en && !interface_object.wr_en) |=> (count == $past(count) - 4'b0001 ) ;
    endproperty
    decrement_assertion : assert property(p5);
    decrement_cover : cover property(p5);
```

```
property p6;
@(posedge interface_object.clk) disable iff(!interface_object.rst_n) (count == interface_object.FIFO_DEPTH) |-> interface_object.full ;
endproperty
full_assertion : assert property(p6);
full_cover : cover property(p6);

property p7;
@(posedge interface_object.clk) disable iff(!interface_object.rst_n) (count == 0) |-> interface_object.empty ;
endproperty
empty_assertion : assert property(p7);
empty_cover : cover property(p7);

property p8;
@(posedge interface_object.clk) disable iff(!interface_object.rst_n) (count == interface_object.FIFO_DEPTH-1) |-> interface_object.almostfull ;
endproperty
almostfull_assertion : assert property(p8);
almostfull_cover : cover property(p8);

property p9;
@(posedge interface_object.clk) disable iff(!interface_object.rst_n) (count == 1) |-> interface_object.almostempty ;
endproperty
almostempty_assertion : assert property(p9);
almostempty_cover : cover property(p9);

//`endif

endmodule
```

Coverage ➔ with bugs:

//the full coverage report will be found in the .rar file

FIFO_project > fcover_report.txt

```
Coverage Report by instance with details

================================================================================
=== Instance: /FIFO_coverage_pkg
=== Design Unit: work.FIFO_coverage_pkg
================================================================================

Covergroup Coverage:
    Covergroups                 1        na        na    85.20%
        Coverpoints/Crosses    16        na        na        na
            Covergroup Bins    56        47         9    83.92%
--------------------------------------------------------------------------------------------------------------------
```

FIFO_project > fifocoveragereport.txt

```
Coverage Report by instance with details

================================================================================
=== Instance: /\TOP /interface_object
=== Design Unit: work.FIFO_if
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         86        84         2    97.67%

```

Questa snippets➔with bugs

```
# error counter ==>         141, correct counter ==>          11
```

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
/TOP/fifo_dut/writ...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut...	Concurrent	SVA	on	52	0	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/over...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/incre...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/decr...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/full_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/emp...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut...	Concurrent	SVA	on	12	0	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/almo...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_testbech...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/TOP/fifo_testbech...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/TOP/fifo_testbech...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓

After fixing the bugs:

Top module:

FIFO_project > top.sv

```
module TOP;
    bit clk;
    initial begin
        clk = 0;
        forever begin
            #50 clk =~ clk;
        end
    end
        FIFO_if interface_object(clk);
        FIFO fifo_dut (interface_object);
        FIFO_tb fifo_testbech (interface_object);
        FIFO_monitor fifo_monitor (interface_object);
endmodule
```

RTL design after fixing the bugs:

FIFO_project > fifo_rtl_noBugs.sv

```
module FIFO(FIFO_if.fifo_dut interface_object);

    // declaration of max. FIFO address
    localparam max_fifo_addr = $clog2(interface_object.FIFO_DEPTH);   // max_fifo_addr = 3

    // declaration of Memory (FIFO)
    reg [interface_object.FIFO_WIDTH-1:0] mem [interface_object.FIFO_DEPTH-1:0];

    // Declaration of read & write pointers
    reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
    reg [max_fifo_addr:0] count;             // extra bit to distinguish between full & empty flags & it represents the fill level of the FIFO

    // always block specialized for writing operation
    always @(posedge interface_object.clk or negedge interface_object.rst_n) begin
        if (!interface_object.rst_n) begin
            wr_ptr <= 0;
            interface_object.overflow <= 0;
            interface_object.wr_ack <= 0;            // reset the sequential outputs as wr_ack , overflow
        end
        else if (interface_object.wr_en && count < interface_object.FIFO_DEPTH) begin
            mem[wr_ptr] <= interface_object.data_in;
            interface_object.wr_ack <= 1;
            wr_ptr <= wr_ptr + 1;
        end
        else begin
            interface_object.wr_ack <= 0;
            if (interface_object.full && interface_object.wr_en)
                interface_object.overflow <= 1;
            else
                interface_object.overflow <= 0;
        end
    end
```

```
    // always block specialized for reading operation
    always @(posedge interface_object.clk or negedge interface_object.rst_n) begin
        if (!interface_object.rst_n) begin
            rd_ptr <= 0;
            interface_object.underflow <= 0;
            interface_object.data_out <= 0;       // reset the sequential outputs as data_out , underflow
        end
        else if (interface_object.rd_en && count != 0) begin
            interface_object.data_out <= mem[rd_ptr];
            rd_ptr <= rd_ptr + 1;
            end
        else
        begin
            if(interface_object.empty && interface_object.rd_en)
                interface_object.underflow  <= 1;
            else
                interface_object.underflow  <= 0;
        end
    end

```

```
54      // always block specialized for counter signal
55      always @(posedge interface_object.clk or negedge interface_object.rst_n) begin
56          if (!interface_object.rst_n) begin
57              count <= 0;
58          end
59          else begin
60              if  ( ({interface_object.wr_en, interface_object.rd_en} == 2'b10) && !interface_object.full)
61                  count <= count + 1;
62              else if ( ({interface_object.wr_en, interface_object.rd_en} == 2'b01) && !interface_object.empty)
63                  count <= count - 1;
64              else if (({interface_object.wr_en, interface_object.rd_en} == 2'b11) && interface_object.full)      // priority for write operation
65                  count <= count - 1;
66              else if (({interface_object.wr_en, interface_object.rd_en} == 2'b11) && interface_object.empty)      // priority for read operation
67                  count <= count + 1;
68          end
```

```
70
71      // continous assignment for the combinational outputs
72      assign interface_object.full = (count == interface_object.FIFO_DEPTH)? 1 : 0;
73      assign interface_object.empty = (count == 0)? 1 : 0;
74      assign interface_object.almostfull = (count == interface_object.FIFO_DEPTH-1)? 1 : 0;
75      assign interface_object.almostempty = (count == 1)? 1 : 0;
76
```

```
80      property p1;
81      @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (interface_object.wr_en && !interface_object.full) |=> interface_object.wr_ack ;
82      endproperty
83      write_acknowledge : assert property(p1);
84      write_acknowledge_cover : cover property(p1);
85
86      property p2;
87      @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (interface_object.empty && interface_object.rd_en) |=> interface_object.underflow ;
88      endproperty
89      underflow_assertion : assert property(p2);
90      underflow_cover : cover property(p2);
91
92      property p3;
93      @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (interface_object.full & interface_object.wr_en) |=> interface_object.overflow ;
94      endproperty
95      overflow_assertion : assert property(p3);
96      overflow_cover : cover property(p3);
97
98      property p4;
99      @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (!interface_object.full & interface_object.wr_en & !interface_object.rd_en) |=> (count == $past(count) + 4'b0001 ) ;
100     endproperty
101     increment_assertion : assert property(p4);
102     increment_cover : cover property(p4);
103
104     property p5;
105     @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (!interface_object.empty && interface_object.rd_en && !interface_object.wr_en) |=> (count == $past(count) - 4'b0001 ) ;
106     endproperty
107     decrement_assertion : assert property(p5);
108     decrement_cover : cover property(p5);
109
110     property p6;
111     @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (count == interface_object.FIFO_DEPTH) |-> interface_object.full ;
112     endproperty
113     full_assertion : assert property(p6);
114     full_cover : cover property(p6);
115
```

```
116     property p7;
117     @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (count == 0) |-> interface_object.empty ;
118     endproperty
119     empty_assertion : assert property(p7);
120     empty_cover : cover property(p7);
121
122     property p8;
123     @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (count == interface_object.FIFO_DEPTH-1) |-> interface_object.almostfull ;
124     endproperty
125     almostfull_assertion : assert property(p8);
126     almostfull_cover : cover property(p8);
127
128     property p9;
129     @(posedge interface_object.clk) disable iff(!interface_object.rst_n) (count == 1) |-> interface_object.almostempty ;
130     endproperty
131     almostempty_assertion : assert property(p9);
132     almostempty_cover : cover property(p9);
133
134     //`endif
135
136     endmodule
```

Package 1:

FIFO_project > pkg_1.sv

```
package FIFO_transaction_pkg; //pkg_1
    class FIFO_transaction;
    //parameters:
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;
    //inputs
    rand bit [FIFO_WIDTH-1:0]data_in;
    rand bit rst_n, wr_en, rd_en;
    //outputs
    logic [FIFO_WIDTH-1:0]data_out;
    logic wr_ack, overflow, full, empty, almostfull, almostempty, underflow;
    //the 2 integers WR_EN_ON_DIST and RD_EN_ON_DIST
    int WR_EN_ON_DIST = 70;
    int RD_EN_ON_DIST = 30;

    //constraint rst
    constraint constrained_rst{ rst_n dist{0:/1, 1:/99}; }

    //constraint write and read as required
    constraint constrained_write{ wr_en dist{0:/(100-WR_EN_ON_DIST), 1:/WR_EN_ON_DIST}; }
    constraint constrained_read{ rd_en dist{0:/(100-RD_EN_ON_DIST), 1:/RD_EN_ON_DIST}; }

    //additional constraints =====> write_only and read_only
    constraint write_only{ rst_n == 1; rd_en == 1; wr_en == 1;}
    constraint read_only{ rst_n == 1; rd_en == 1; wr_en == 0;}

    endclass
endpackage
```

Package 2:

FIFO_project > pkg_2.sv

```
package FIFO_coverage_pkg;
    //import the prev pkg(FIFO_transaction_pkg)
    import FIFO_transaction_pkg ::*;

    class FIFO_coverage;
        //object from the FIFO_transaction class
        FIFO_transaction F_cvg_txn = new();
        //function with no return value
        function void sample_data (input FIFO_transaction F_txn);
            F_cvg_txn = F_txn;
            cover_group.sample();
        endfunction

        covergroup cover_group;
            //coverpoints to be used in the cross coverage
            //the coverage needed is cross coverage between 3 signals which are write enable, read enable and each output control signals
            //except outputs except data_out
            //wr_ack, overflow, full, empty, almostfull, almostempty, underflow
            wr_en_cover_point:coverpoint F_cvg_txn.wr_en;
            rd_en_cover_point:coverpoint F_cvg_txn.rd_en;
            wr_ack_cover_point:coverpoint F_cvg_txn.wr_ack;
            overflow_cover_point:coverpoint F_cvg_txn.overflow;
            full_cover_point:coverpoint F_cvg_txn.full;
            empty_cover_point:coverpoint F_cvg_txn.empty;
            almostfull_cover_point:coverpoint F_cvg_txn.almostfull;
            almostempty_cover_point:coverpoint F_cvg_txn.almostempty;
            underflow_cover_point:coverpoint F_cvg_txn.underflow;
            //cross coverage to make sure that all combinations of (write and read enable) took place
            //in all state of the FIFO ==> with all o/p except d_out.
            //then we make 7 cross coverages(7 outputs, each o/p indivdually with both the wr_en and rd_en)
            //since, all coverpoints are 1 bit the bins will have only two states 0 or 1
            //bin{1} will be the high state
```

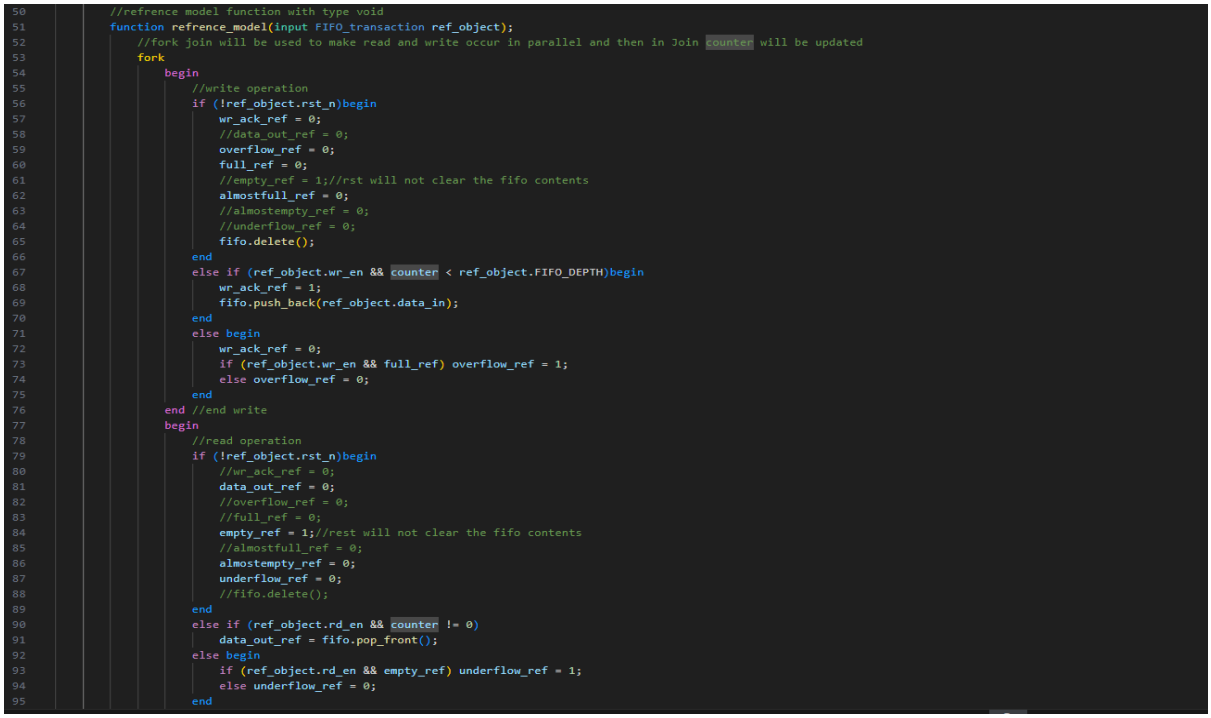
```
            wr_en_rd_en_wr_ack:cross wr_en_cover_point, rd_en_cover_point, wr_ack_cover_point {
                ignore_bins wr_en_low_wr_ack_high = !binsof(wr_en_cover_point) intersect{1} && binsof(wr_ack_cover_point) intersect{1};
                ignore_bins read_write_active_with_wr_ack = ! binsof(wr_en_cover_point) intersect {1} && binsof(rd_en_cover_point) intersect {1} && binsof(wr_ack_cover_point) intersect {1};
            }
            wr_en_rd_en_overflow:cross wr_en_cover_point, rd_en_cover_point, overflow_cover_point {
                ignore_bins wr_en_low_overflow_high = !binsof(wr_en_cover_point) intersect{1} && binsof(overflow_cover_point) intersect{1};
            }
            wr_en_rd_en_full:cross wr_en_cover_point, rd_en_cover_point, full_cover_point {
                ignore_bins wr_en_low_full_high = !binsof(wr_en_cover_point) intersect{1} && binsof(full_cover_point) intersect{1};
                ignore_bins rd_en_high_full_high = binsof(rd_en_cover_point) intersect{1} && binsof(full_cover_point) intersect{1};
            }
            wr_en_rd_en_empty:cross wr_en_cover_point, rd_en_cover_point, empty_cover_point{
                ignore_bins rd_en_low_empty_high = !binsof(rd_en_cover_point) intersect{1} && binsof(empty_cover_point) intersect{1};
                ignore_bins wr_en_low_empty_high = !binsof(wr_en_cover_point) intersect{1} && binsof(empty_cover_point) intersect{1};
            }
            wr_en_rd_en_almostfull:cross wr_en_cover_point, rd_en_cover_point, almostfull_cover_point{
                ignore_bins wr_en_low_almostfull_high = !binsof(wr_en_cover_point) intersect{1} && binsof(almostfull_cover_point) intersect{1};
            }
            wr_en_rd_en_almostempty:cross wr_en_cover_point, rd_en_cover_point, almostempty_cover_point{
                ignore_bins rd_en_low_almostempty_high = !binsof(rd_en_cover_point) intersect{1} && binsof(almostempty_cover_point) intersect{1};
                ignore_bins we_en_low_rd_en_high_almostempty_high = !binsof(wr_en_cover_point) intersect{1} && binsof(rd_en_cover_point) intersect{1} && binsof(almostempty_cover_point) intersect{1};

            }
            wr_en_rd_en_underflow:cross wr_en_cover_point, rd_en_cover_point, underflow_cover_point {
                ignore_bins rd_en_low_underflow_high = !binsof(rd_en_cover_point) intersect{1} && binsof(underflow_cover_point) intersect{1};
                ignore_bins wr_en_high_underflow_high = !binsof(wr_en_cover_point) intersect{1} && binsof(underflow_cover_point) intersect{1};
            }
        endgroup
```

```
        function new();
            cover_group = new();
            //F_cvg_txn = new();
        endfunction
    endclass
endpackage
```

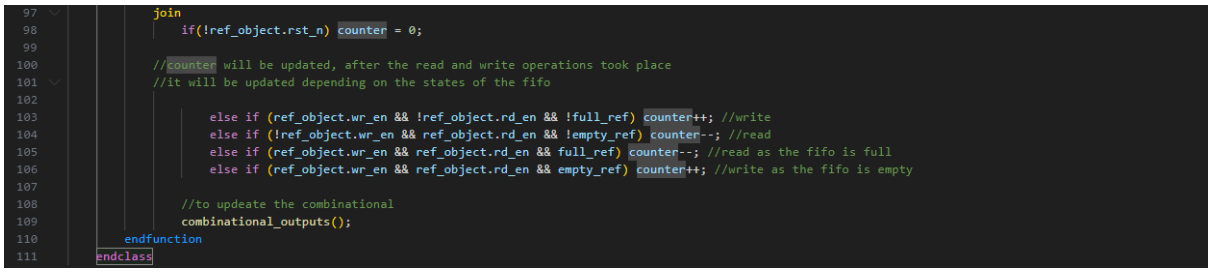
Package 3:

```
FIFO_project > pkg_3.sv
package FIFO_scoreboard_pkg;
    //import the prev pkg(FIFO_transaction_pkg)
    import FIFO_transaction_pkg ::*;
    //import the shared pkg
    import shared_pkg ::*;

    class FIFO_scoreboard;
        //parameters:
        parameter FIFO_WIDTH = 16;
        parameter FIFO_DEPTH = 8;
        //outputs
        bit [FIFO_WIDTH-1:0]data_out_ref;
        bit wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;

        //counter that will be used in determining full, empty, almostfull, almostempty in the queue
        int counter;
        bit [FIFO_WIDTH-1:0] fifo[$]; //$ ==> means that this is a dynamic array which can be modified during runtime
        //object from the FIFO_transaction class
        FIFO_transaction dut_object = new();
        //function to assign the combinational outputs of type void
        //full, empty, almostfull, almostempty ==> the refrence model
        function void combinational_outputs();
            full_ref = (counter == FIFO_DEPTH)? 1:0;
            empty_ref = (counter == 0)? 1:0;
            almostfull_ref = (counter == FIFO_DEPTH-1)? 1:0;
            almostempty_ref = (counter == 1)? 1:0;
        endfunction
```

```
        //check data function to compare the dut and ref models
        function void check_data(input FIFO_transaction dut_object);

            //for simplicity in comparing dut and ref , the flags are concatenated
            logic[6:0] dut_flags;
            logic[6:0] ref_flags;

            refrence_model(dut_object);
            //wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref
            dut_flags = {dut_object.wr_ack, dut_object.overflow, dut_object.full, dut_object.empty, dut_object.almostfull, dut_object.almostempty, dut_object.underflow};
            ref_flags = {wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref};
            if ((dut_object.data_out != data_out_ref) || (dut_flags != ref_flags)) begin
                $display("DUT is NOT EQUAL REF AT TIME = %t", $time);
                error_count++;
            end
            else begin
                correct_count++;
                $display( "WELL DONE, The FIFO = %p",fifo);
            end
        endfunction
```



```
        //refrence model function with type void
        function refrence_model(input FIFO_transaction ref_object);
            //fork join will be used to make read and write occur in parallel and then in Join counter will be updated
            fork
                begin
                    //write operation
                    if (!ref_object.rst_n)begin
                        wr_ack_ref = 0;
                        //data_out_ref = 0;
                        overflow_ref = 0;
                        full_ref = 0;
                        //empty_ref = 1;//rst will not clear the fifo contents
                        almostfull_ref = 0;
                        //almostempty_ref = 0;
                        //underflow_ref = 0;
                        fifo.delete();
                    end
                    else if (ref_object.wr_en && counter < ref_object.FIFO_DEPTH)begin
                        wr_ack_ref = 1;
                        fifo.push_back(ref_object.data_in);
                    end
                    else begin
                        wr_ack_ref = 0;
                        if (ref_object.wr_en && full_ref) overflow_ref = 1;
                        else overflow_ref = 0;
                    end
                end //end write
                begin
                    //read operation
                    if (!ref_object.rst_n)begin
                        //wr_ack_ref = 0;
                        data_out_ref = 0;
                        //overflow_ref = 0;
                        //full_ref = 0;
                        empty_ref = 1;//rest will not clear the fifo contents
                        //almostfull_ref = 0;
                        almostempty_ref = 0;
                        underflow_ref = 0;
                        //fifo.delete();
                    end
                    else if (ref_object.rd_en && counter != 0)
                        data_out_ref = fifo.pop_front();
                    else begin
                        if (ref_object.rd_en && empty_ref) underflow_ref = 1;
                        else underflow_ref = 0;
                    end
```



```
            join
                if(!ref_object.rst_n) counter = 0;

            //counter will be updated, after the read and write operations took place
            //it will be updated depending on the states of the fifo

                    else if (ref_object.wr_en && !ref_object.rd_en && !full_ref) counter++; //write
                    else if (!ref_object.wr_en && ref_object.rd_en && !empty_ref) counter--; //read
                    else if (ref_object.wr_en && ref_object.rd_en && full_ref) counter--; //read as the fifo is full
                    else if (ref_object.wr_en && ref_object.rd_en && empty_ref) counter++; //write as the fifo is empty

                //to updeate the combinational
                combinational_outputs();
        endfunction
    endclass
```

Package 4:

FIFO_project > pkg_4.sv

```
package shared_pkg;
    int error_count, correct_count;
    bit test_finished;
endpackage
```

Interface:

FIFO_project > interface.sv

```
interface FIFO_if(clk);
    input bit clk;
    //interface includes all ports

    //parameters:
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;
    //inputs
    logic [FIFO_WIDTH-1:0]data_in;
    logic  rst_n, wr_en, rd_en;
    //outputs
    logic [FIFO_WIDTH-1:0]data_out;
    logic wr_ack, overflow, full, empty, almostfull, almostempty, underflow;
    //modport dut
    modport fifo_dut (
    input clk, data_in, rst_n, wr_en, rd_en,
    output data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow
    );
    //modport tb=>
    modport fifo_testbech (
    input clk, data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow,
    output data_in, rst_n, wr_en, rd_en
    );
    //modport monitor=>
    modport fifoMonitor (input clk, data_in, rst_n, wr_en, rd_en, data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow);


endinterface
```

Monitor:

FIFO_project > monitor.sv

```
import FIFO_transaction_pkg ::*;
import FIFO_scoreboard_pkg ::*;
import FIFO_coverage_pkg ::*;
import shared_pkg ::*;

module FIFO_monitor(FIFO_if.fifoMonitor interface_object);
    //object from each class
    FIFO_transaction trans_object;
    FIFO_scoreboard score_object;
    FIFO_coverage cove_object;
    initial begin
        trans_object = new();
        score_object = new();
        cove_object = new();
        forever begin
            @(negedge interface_object.clk); //interface clk
            //sample the data of the interface and assign it to the data of the object of class FIFO_transaction
            //data_in, wr_en, rd_en, rst_n, full, empty, almostfull, almostempty, wr_ack, overflow, underflow, data_out
            trans_object.data_in = interface_object.data_in;
            trans_object.wr_en = interface_object.wr_en;
            trans_object.rd_en = interface_object.rd_en;
            trans_object.rst_n = interface_object.rst_n;
            trans_object.full = interface_object.full;
            trans_object.empty = interface_object.empty;
            trans_object.almostfull = interface_object.almostfull;
            trans_object.almostempty = interface_object.almostempty;
            trans_object.wr_ack = interface_object.wr_ack;
            trans_object.overflow = interface_object.overflow;
            trans_object.underflow = interface_object.underflow;
            trans_object.data_out = interface_object.data_out;

            fork
                begin //first one is calling a method named sample_data of the object of class FIFO_coverage
                    cove_object.sample_data(trans_object);
                end
                begin //second process is calling a method named check_data of the object of class FIFO_scoreboard
                    @(posedge interface_object.clk) //checking should happen at the posedge
                    //delay of 20 ms to gaurantee that the o/p is changed at the posedge before checking
                    #20;
                    score_object.check_data(trans_object);
                end
            join
                if(test_finished) begin
                    $display("final values are ==> %p", score_object.fifo);
                    $display("error counter ==> %d, correct counter ==> %d", error_count, correct_count);
                    $stop;
                end
        end
    end
endmodule
```

Testbench:

FIFO_project > testbench.sv

```
import FIFO_transaction_pkg ::*;
// import FIFO_scoreboard_pkg ::*;
// import FIFO_coverage_pkg ::*;
import shared_pkg ::*;
module FIFO_tb (FIFO_if.fifo_testbech interface_object);
    //create an object from the fifo_transaction class to randomize
    FIFO_transaction tb_object;
    initial begin
        tb_object = new();

        interface_object.rst_n = 0;
        @(negedge interface_object.clk);
        @(negedge interface_object.clk);
        //check write only, disable all constarints except that of write_only
        interface_object.rst_n = 1;
        tb_object.constraint_mode(0);
        tb_object.write_only.constraint_mode(1);
        //randomize inputs ==> data_in, wr_en, rd_en, rst_n
        repeat(50)begin
            assert(tb_object.randomize());
            interface_object.data_in = tb_object.data_in;
            interface_object.wr_en = tb_object.wr_en;
            interface_object.rd_en = tb_object.rd_en;
            interface_object.rst_n = tb_object.rst_n;
            @(negedge interface_object.clk);
        end

        //check read only, disable all constarints except that of read_only
        tb_object.constraint_mode(0);
        tb_object.read_only.constraint_mode(1);
        repeat(50)begin
            assert(tb_object.randomize());
            interface_object.data_in = tb_object.data_in;
            interface_object.wr_en = tb_object.wr_en;
            interface_object.rd_en = tb_object.rd_en;
            interface_object.rst_n = tb_object.rst_n;
        end

        //check for write and read , disable constarints  of read_only and write_only and enable others
        tb_object.constraint_mode(1);
        tb_object.write_only.constraint_mode(0);
        tb_object.read_only.constraint_mode(0);
        repeat(100)begin
            assert(tb_object.randomize());
            interface_object.data_in = tb_object.data_in;
            interface_object.wr_en = tb_object.wr_en;
            interface_object.rd_en = tb_object.rd_en;
            interface_object.rst_n = tb_object.rst_n;
            @(negedge interface_object.clk);
        end
        //At the end of the test, the tb will assert a signal named test_finished
        test_finished = 1;
    end
endmodule
```

Do file:

FIFO_project > run_FIFO.do

```
vlog -f src_files.list -mfcu +define+SIM +cover
vsim -voptargs=+acc work.TOP -cover
add wave *
coverage save fifocoveragereport.ucdb -onexit -du work.TOP
run -all
coverage report -detail -cvg -directive -comments -output fcover_report.txt /FIFO_coverage_pkg/FIFO_coverage/cover_group
#quit -sim
vcover report fifocoveragereport.ucdb -details -annotate -all -output fifocoveragereport.txt

```

Src_files.list:

FIFO_project > src_files.list

```
pkg_4.sv
pkg_1.sv
pkg_2.sv
pkg_3.sv
fifo_rtl_noBugs.sv
interface.sv
monitor.sv
testbench.sv
top.sv
```

Some of Questa snippets:

```
# error counter ==>           0, correct counter ==>         152
```

```
# WELL DONE, The FIFO = '{51082, 29829, 37721}
# WELL DONE, The FIFO = '{51082, 29829, 37721, 20300}
# WELL DONE, The FIFO = '{51082, 29829, 37721, 20300}
# WELL DONE, The FIFO = '{51082, 29829, 37721, 20300, 17834}
# WELL DONE, The FIFO = '{51082, 29829, 37721, 20300, 17834, 63099}
# WELL DONE, The FIFO = '{51082, 29829, 37721, 20300, 17834, 63099}
# WELL DONE, The FIFO = '{51082, 29829, 37721, 20300, 17834, 63099}
# WELL DONE, The FIFO = '{29829, 37721, 20300, 17834, 63099, 35717}
# WELL DONE, The FIFO = '{29829, 37721, 20300, 17834, 63099, 35717, 47571}
# WELL DONE, The FIFO = '{29829, 37721, 20300, 17834, 63099, 35717, 47571}
# WELL DONE, The FIFO = '{29829, 37721, 20300, 17834, 63099, 35717, 47571}
# WELL DONE, The FIFO = '{29829, 37721, 20300, 17834, 63099, 35717, 47571, 44223}
# WELL DONE, The FIFO = '{37721, 20300, 17834, 63099, 35717, 47571, 44223}
# WELL DONE, The FIFO = '{37721, 20300, 17834, 63099, 35717, 47571, 44223, 41132}
# WELL DONE, The FIFO = '{37721, 20300, 17834, 63099, 35717, 47571, 44223, 41132}
# WELL DONE, The FIFO = '{37721, 20300, 17834, 63099, 35717, 47571, 44223, 41132}
# WELL DONE, The FIFO = '{20300, 17834, 63099, 35717, 47571, 44223, 41132}
# WELL DONE, The FIFO = '{20300, 17834, 63099, 35717, 47571, 44223, 41132, 61290}
# WELL DONE, The FIFO = '{20300, 17834, 63099, 35717, 47571, 44223, 41132, 61290}
# WELL DONE, The FIFO = '{20300, 17834, 63099, 35717, 47571, 44223, 41132, 61290}
```

```
# WELL DONE, The FIFO = '{31950}
# WELL DONE, The FIFO = '{54164}
# WELL DONE, The FIFO = '{61019}
# WELL DONE, The FIFO = '{37750}
# WELL DONE, The FIFO = '{23018}
# WELL DONE, The FIFO = '{56936}
# WELL DONE, The FIFO = '{19112}
# WELL DONE, The FIFO = '{30214}
# WELL DONE, The FIFO = '{29687}
# WELL DONE, The FIFO = '{55586}
# WELL DONE, The FIFO = '{52846}
# WELL DONE, The FIFO = '{54785}
# WELL DONE, The FIFO = '{18859}
# WELL DONE, The FIFO = '{38847}
# WELL DONE, The FIFO = '{36683}
```

Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
/TOP/fifo_dut/writ...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/und...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/over...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/incre...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/decr...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/full_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/emp...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/almo...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_dut/almo...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge interface_objec...	✓
/TOP/fifo_testbech...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/TOP/fifo_testbech...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/TOP/fifo_testbech...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓

Coverage: ➔after fixing the bugs:

//the full code and functional coverage are included in the .rar file

//I failed to make the coverage 100%

FIFO_project > fcover_report.txt

```
Coverage Report by instance with details

================================================================================
=== Instance: /FIFO_coverage_pkg
=== Design Unit: work.FIFO_coverage_pkg
================================================================================

Covergroup Coverage:
    Covergroups                     1        na        na    98.75%
        Coverpoints/Crosses        16        na        na        na
            Covergroup Bins        56        55         1    98.21%
```

FIFO_project > fifocoveragereport.txt

```
Coverage Report by instance with details

================================================================================
=== Instance: /\TOP /interface_object
=== Design Unit: work.FIFO_if
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         86        82         4    95.34%
```