SV Project - Synchronous FIFO

By

Omar Ahmed Ragab

FIFO Testing Project

Codes:

Top:

```
Goto  Tools  Project  Preferences  Help
module FIFO_top ();
    bit clk;
    always #1 clk = ~clk;
    FIFO_if V_if(clk);
    FIFO DUT(V_if);
    FIFO_tb tb(V_if);
    monitor_fifo mon(V_if);
endmodule
```

Interface:

```
Goto  Tools  Project  Preferences  Help
import shared_pkg::*;
interface FIFO_if (clk);
    logic [FIFO_WIDTH-1:0] data_in;
    input clk;
    bit rst_n, wr_en, rd_en;
    logic [FIFO_WIDTH-1:0] data_out;
    logic wr_ack, overflow,underflow;
    bit full, empty, almostfull, almostempty;
modport TEST (output data_in, rst_n, wr_en, rd_en, input clk, data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow);
modport DUT (input data_in, rst_n, wr_en, rd_en, clk, output data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow);
modport MONITOR (input clk, data_in, rst_n, wr_en, rd_en, data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow);

endinterface : FIFO_if
```

Shared package:

```
na\Project1\shared_pkg.sv (Project1) - Sublime Text (UNREGISTERED)
ew  Goto  Tools  Project  Preferences  Help
package shared_pkg;
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;
    bit test_finished = 0;
    int error_count = 0;
    int correct_count = 0;
    int iteration = 0;
endpackage : shared_pkg
```

Design:

```
Goto  Tools  Project  Preferences  Help
1   ////////////////////////////////////////////////////////////////////////////////
2   // Author: Kareem Waseem
3   // Course: Digital Verification using SV & UVM
4   //
5   // Description: FIFO Design
6   //
7   ////////////////////////////////////////////////////////////////////////////////
8   import shared_pkg::*;
9   module FIFO(FIFO_if.DUT V_if);
10
11  localparam max_fifo_addr = $clog2(FIFO_DEPTH);
12
13  reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];
14
15  reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
16  reg [max_fifo_addr:0] count;
17
18  always @(posedge V_if.clk or negedge V_if.rst_n) begin
19      if (!V_if.rst_n) begin
20          wr_ptr <= 0;
21          //reset is not complete
22          V_if.wr_ack <=0;
23          V_if.overflow <= 0;
24      end
25      else if (V_if.wr_en && count < FIFO_DEPTH) begin
26          mem[wr_ptr] <= V_if.data_in;
27          V_if.wr_ack <= 1;
28          wr_ptr <= wr_ptr + 1;
29      end
30      else begin
31          V_if.wr_ack <= 0;
32          if (V_if.full & V_if.wr_en)
33              V_if.overflow <= 1;
34          else
35              V_if.overflow <= 0;
36      end
37  end
```

```
39  always @(posedge V_if.clk or negedge V_if.rst_n) begin
40      if (!V_if.rst_n) begin
41          rd_ptr <= 0;
42          //underflow reset
43          V_if.underflow <= 0;
44      end
45      else if (V_if.rd_en && count != 0) begin
46          V_if.data_out <= mem[rd_ptr];
47          rd_ptr <= rd_ptr + 1;
48      end
49      //handling the underflow signal
50      else begin
51          if (V_if.empty & V_if.rd_en)
52              V_if.underflow <= 1;
53          else
54              V_if.underflow <= 0;
55      end
56  end
57
58  always @(posedge V_if.clk or negedge V_if.rst_n) begin
59      if (!V_if.rst_n) begin
60          count <= 0;
61      end
62      else begin
63          case ({V_if.wr_en,V_if.rd_en})
64              2'b00 : begin
65                  V_if.data_out <= V_if.data_out;
66              end
67              2'b01 : begin
68                  if(!V_if.empty)
69                      count <= count - 1;
70              end
71              2'b10 : begin
72                  if(!V_if.full)
73                      count <= count + 1;
74              end
75              2'b11 : begin
76                  if(V_if.full)
77                      count <= count - 1;
78                  else if(V_if.empty)
79                      count <= count + 1;
80                  else
81                      count <= count;
82              end
83          endcase
84          // if   ( ({V_if.wr_en, V_if.rd_en} == 2'b10) && !V_if.full)
85          //  count <= count + 1;
86          // else if ( ({V_if.wr_en, V_if.rd_en} == 2'b01) && !V_if.empty)
87          //  count <= count - 1;
88      end
89  end
90
91  assign V_if.full = (count == FIFO_DEPTH)? 1 : 0;
92  assign V_if.empty = (count == 0)? 1 : 0;
93  // assign V_if.underflow = (count == 0 && V_if.rd_en)? 1 : 0;
94  assign V_if.almostfull = (count == FIFO_DEPTH-1)? 1 : 0;
95  assign V_if.almostempty = (count == 1)? 1 : 0;
```

Assertions in main:

```
`ifdef SVA
always_comb begin
    if(!V_if.rst_n)
        a_reset: assert final(V_if.full == 0 && V_if.empty == 1 && V_if.almostfull == 0 && V_if.almostempty == 0 &&
                              V_if.overflow == 0 && V_if.underflow == 0 && count == 0 && V_if.wr_ack == 0);
end

property reset_reg;
    @(posedge V_if.clk) (V_if.rst_n == 0) |=> ( V_if.overflow == 0 && V_if.underflow == 0 && count == 0 && V_if.wr_ack == 0);
endproperty

property full_prop;
    @(posedge V_if.clk) disable iff(!V_if.rst_n) (count == FIFO_DEPTH) |-> (V_if.full == 1);
endproperty

property almostfull_prop;
    @(posedge V_if.clk) disable iff(!V_if.rst_n) (count == FIFO_DEPTH-1) |-> (V_if.almostfull == 1);
endproperty

property empty_prop;
    @(posedge V_if.clk) (count == 0) |-> (V_if.empty == 1);
endproperty

property almostempty_prop;
    @(posedge V_if.clk) disable iff(!V_if.rst_n) (count == 1) |-> (V_if.almostempty == 1);
endproperty

property overflow_prop;
    @(posedge V_if.clk) disable iff(!V_if.rst_n) ((count == FIFO_DEPTH) && V_if.wr_en == 1) |=> (V_if.overflow == 1);
endproperty

property underflow_prop;
    @(posedge V_if.clk) disable iff(!V_if.rst_n) ((count == 0) && V_if.rd_en == 1) |=> (V_if.underflow == 1);
endproperty

property wr_ack_prop;
    @(posedge V_if.clk) disable iff(!V_if.rst_n) (V_if.wr_en && (count < FIFO_DEPTH)) |=> (V_if.wr_ack== 1);
endproperty


assert property(reset_reg); cover property(reset_reg);
assert property(full_prop); cover property(full_prop);
assert property(almostfull_prop); cover property(almostfull_prop);
assert property(empty_prop); cover property(empty_prop);
assert property(almostempty_prop); cover property(almostempty_prop);
assert property(overflow_prop); cover property(overflow_prop);
assert property(underflow_prop); cover property(underflow_prop);
assert property(wr_ack_prop); cover property(wr_ack_prop);

`endif

endmodule
```

Run file:

```
View  Goto  Tools  Project  Preferences  Help
vlib work
vlog -f src_files.list  +cover +define +SVA
vsim -voptargs=+acc FIFO_top -cover
add wave -position insertpoint sim:/FIFO_top/V_if/*
add wave -position insertpoint  \
sim:/shared_pkg::iteration
coverage save FIFO_TOP.ucdb -onexit -du work.FIFO_top
run -all
```

Monitor:

\Project1\FIFO_monitor.sv • (Project1) - Sublime Text (UNREGISTERED)

Goto Tools Project Preferences Help

```
import FIFO_Transaction_pkg::*;
import FIFO_scoreboard_pkg::*;
import FIFO_coverage_pkg::*;
import shared_pkg::*;
module monitor_fifo(FIFO_if.MONITOR V_if);
  FIFO_transaction fifo_txn;
  FIFO_coverage fifo_cov;
  FIFO_scoreboard fifo_sb;
  initial begin
    fifo_txn = new();
    fifo_cov = new();
    fifo_sb = new();
    forever begin
      @(negedge V_if.clk);
      fifo_txn.rst_n     = V_if.rst_n;
      fifo_txn.data_in     = V_if.data_in;
      fifo_txn.wr_en       = V_if.wr_en;
      fifo_txn.rd_en       = V_if.rd_en;
      fifo_txn.data_out    = V_if.data_out;
      fifo_txn.wr_ack      = V_if.wr_ack;
      fifo_txn.full        = V_if.full;
      fifo_txn.empty       = V_if.empty;
      fifo_txn.almostfull  = V_if.almostfull;
      fifo_txn.almostempty = V_if.almostempty;
      fifo_txn.overflow    = V_if.overflow;
      fifo_txn.underflow   = V_if.underflow;

      fork
        begin
          fifo_cov.sample_data(fifo_txn);
        end

        begin
          fifo_sb.check_data(fifo_txn);
        end
      join

      if (test_finished) begin
        $display("Simulation finished!");
        $display("Correct transactions: %0d",correct_count);
        $display("Errors found: %0d",error_count);
        $stop;
        $finish;
      end
    end
  end

endmodule
```

Testbench:

```
Goto  Tools  Project  Preferences  Help
import shared_pkg::*;
import FIFO_Transaction_pkg::*;
module FIFO_tb(FIFO_if.TEST V_if);

  FIFO_transaction trans = new();
int i = 0;
  initial begin
    V_if.rst_n = 1;
    @(negedge V_if.clk); #0;
    V_if.rst_n = 0;
    @(negedge V_if.clk); #0;
    V_if.rst_n = 1;
    @(negedge V_if.clk); #0;
    V_if.wr_en = 1;
    V_if.data_in = 6;
    V_if.rd_en = 1;
    @(negedge V_if.clk); #0;
    V_if.rd_en = 0;
    for(int i = 0;i<10;i++)begin
      V_if.wr_en = 1;
      V_if.data_in = i;
      @(negedge V_if.clk); #0;
    end
    V_if.wr_en = 0;
    @(negedge V_if.clk); #0;
    V_if.wr_en = 1;
    V_if.rd_en = 1;
    @(negedge V_if.clk); #0;
    for(int i = 10;i<22;i++)begin
      V_if.rd_en = 1;
      V_if.data_in = i;
      @(negedge V_if.clk); #0;
    end
    V_if.wr_en = 0;
    V_if.rd_en = 0;
    @(negedge V_if.clk); #0;
    V_if.wr_en = 1;
    V_if.rd_en = 1;
    @(negedge V_if.clk); #0;
    for(int i = 0;i<32768;i++) begin
    assert(trans.randomize());
        V_if.rst_n  = trans.rst_n;
        V_if.wr_en = trans.wr_en;
        V_if.rd_en     = trans.rd_en;
        V_if.data_in = trans.data_in;
        @(negedge V_if.clk); #0;
    end
    test_finished = 1;
  end
endmodule
```

Transaction package:

```
Goto  Tools  Project  Preferences  Help
package FIFO_Transaction_pkg;
  import shared_pkg::*;
  class FIFO_transaction;

    rand bit rst_n;
    randc bit wr_en;
    randc bit rd_en;
    randc logic [15:0] data_in;
    logic [15:0] data_out;
    bit wr_ack;
    bit overflow;
    bit full, empty, almostfull, almostempty, underflow;

    int RD_EN_ON_DIST;
    int WR_EN_ON_DIST;

    function new(int rd_dist = 30, int wr_dist = 70);
      this.RD_EN_ON_DIST = rd_dist;
      this.WR_EN_ON_DIST = wr_dist;
    endfunction

    constraint reset_assertion {
      rst_n dist {1'b1 :/ 99, 1'b0 :/ 1};
    }

    constraint write_enable_distribution {
      wr_en dist {1'b1 :/ WR_EN_ON_DIST, 1'b0 :/ (100 - WR_EN_ON_DIST)};
    }

    constraint read_enable_distribution {
      rd_en dist {1'b1 :/ RD_EN_ON_DIST, 1'b0 :/ (100 - RD_EN_ON_DIST)};
    }
  endclass

endpackage
```

Scoreboard package:

```
Goto  Tools  Project  Preferences  Help
package FIFO_scoreboard_pkg;
  import FIFO_Transaction_pkg::*;
  import shared_pkg::*;
  class FIFO_scoreboard;
    logic [FIFO_WIDTH-1:0] data_out_ref;
    bit full_ref, empty_ref, almostfull_ref, almostempty_ref, overflow_ref, underflow_ref, wr_ack_ref;
    static logic [FIFO_WIDTH-1:0] fifo_queue[$];
    static int count_ref=0;

    function void check_data(FIFO_transaction F_txn);
      iteration++;
      reference_model(F_txn);

      if (F_txn.data_out !== data_out_ref) begin
        error_count++;
        $display("******************ERROR: Mismatch detected in FIFO transaction in iteration number=%0d.",iteration);
      end else begin
        correct_count++;
      end
      $display("Expected: data_out_ref=%0d, full_ref=%0b, empty_ref=%0b, almostfull_ref=%0b, almostempty_ref=%0b, overflow_ref=%0b, underflow_ref=%0b, wr_ack_ref=%0b;",
          data_out_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, overflow_ref, underflow_ref, wr_ack_ref);

      $display("Got: data_out=%0d, full=%0b, empty=%0b, almostfull=%0b, almostempty=%0b, overflow=%0b, underflow=%0b, wr_ack=%0b",
        F_txn.data_out, F_txn.full, F_txn.empty, F_txn.almostfull, F_txn.almostempty, F_txn.overflow, F_txn.underflow, F_txn.wr_ack);

      $display("queue size=%0d",fifo_queue.size());

      for (int i = 0; i < fifo_queue.size(); i++) begin
          $display("Element at index %0d: %0d", i, fifo_queue[i]);
      end
    endfunction
```

Goto Tools Project Preferences Help

```
    function void reference_model(FIFO_transaction F_txn);
        wr_ack_ref = 0;
        overflow_ref = 0;
        underflow_ref = 0;
        if (!F_txn.rst_n) begin
          fifo_queue.delete();
          count_ref = 0;
          //data_out_ref = F_txn.data_out;
        wr_ack_ref = 0;
        overflow_ref = 0;
        underflow_ref = 0;
        end
        else begin
          case ({F_txn.wr_en,F_txn.rd_en})
            2'b10: begin
              if(fifo_queue.size() < FIFO_DEPTH) begin
                fifo_queue.push_back(F_txn.data_in);
                wr_ack_ref = 1;
                overflow_ref = 0;
              end
              else
                overflow_ref = 1;
            end
            2'b01 : begin
              if(fifo_queue.size() > 0) begin
                data_out_ref = fifo_queue.pop_front();
                underflow_ref = 0;
              end
              else
                underflow_ref = 1;
            end
            2'b00 : begin
              //data_out_ref = F_txn.data_out;
            end
            2'b11 : begin
              if(fifo_queue.size() < FIFO_DEPTH && fifo_queue.size() > 0) begin
                fifo_queue.push_back(F_txn.data_in);
                wr_ack_ref = 1;
                data_out_ref = fifo_queue.pop_front();
              end
              else if(fifo_queue.size() == FIFO_DEPTH) begin
                data_out_ref = fifo_queue.pop_front();
              end
              else if(fifo_queue.size() == 0) begin
                fifo_queue.push_back(F_txn.data_in);
                wr_ack_ref = 1;
              end
            end
          endcase
        end
          full_ref = (fifo_queue.size() == FIFO_DEPTH);
          almostfull_ref = (fifo_queue.size() == FIFO_DEPTH - 1);
          empty_ref = (fifo_queue.size() == 0);
          almostempty_ref = (fifo_queue.size() == 1);
    endfunction
  endclass
endpackage
```

Coverage package:

```
package FIFO_coverage_pkg;
  import FIFO_Transaction_pkg::*;
  class FIFO_coverage;
    FIFO_transaction F_cvg_txn;

    covergroup fifo_cg;
      cp_we_en:coverpoint F_cvg_txn.wr_en;
      cp_rd_en:coverpoint F_cvg_txn.rd_en;
      cp_wr_ack_signal: coverpoint F_cvg_txn.wr_ack;
      cp_overflow_signal: coverpoint F_cvg_txn.overflow;
      cp_full_signal: coverpoint F_cvg_txn.full;
      cp_empty_signal: coverpoint F_cvg_txn.empty;
      cp_almostfull_signal: coverpoint F_cvg_txn.almostfull;
      cp_almostempty_signal: coverpoint F_cvg_txn.almostempty;
      cp_underflow_signal: coverpoint F_cvg_txn.underflow;

      wr_en_cross_rd_en: cross cp_we_en, cp_rd_en;

      wr_en_cross_wr_ack: cross cp_we_en, cp_wr_ack_signal {
            illegal_bins illegal_wr_ack_on_no_wr = binsof(cp_we_en) intersect{0} && binsof(cp_wr_ack_signal) intersect{1};}
      wr_en_cross_overflow: cross cp_we_en, cp_overflow_signal {
            illegal_bins illegal_overflow_on_no_wr = binsof(cp_we_en) intersect{0} && binsof(cp_overflow_signal) intersect{1};
            }
      wr_en_cross_full: cross cp_we_en, cp_full_signal;
      wr_en_cross_empty: cross cp_we_en, cp_empty_signal;
      wr_en_cross_almostfull: cross cp_we_en, cp_almostfull_signal;
      wr_en_cross_almostempty: cross cp_we_en, cp_almostempty_signal;
      wr_en_cross_underflow: cross cp_we_en, cp_underflow_signal;

      rd_en_cross_wr_ack: cross cp_rd_en, cp_wr_ack_signal;
      rd_en_cross_overflow: cross cp_rd_en, cp_overflow_signal {
            illegal_bins illegal_overflow_on_rd = binsof(cp_rd_en) intersect{1} && binsof(cp_overflow_signal) intersect{1};
            }
      rd_en_cross_full: cross cp_rd_en, cp_full_signal {
            illegal_bins illegal_full_on_rd = binsof(cp_rd_en) intersect{1} && binsof(cp_full_signal) intersect{1};
            }
      rd_en_cross_empty: cross cp_rd_en, cp_empty_signal;
      rd_en_cross_almostfull: cross cp_rd_en, cp_almostfull_signal;
      rd_en_cross_almostempty: cross cp_rd_en, cp_almostempty_signal;
      rd_en_cross_underflow: cross cp_rd_en, cp_underflow_signal {
            illegal_bins illegal_underflow_on_rd = binsof(cp_rd_en) intersect{0} && binsof(cp_underflow_signal) intersect{1};
            }

    endgroup
    function new();
      this.fifo_cg = new();
    endfunction

    function void sample_data(FIFO_transaction F_txn);
      F_cvg_txn = F_txn;
      fifo_cg.sample();
    endfunction
  endclass
endpackage
```

Bugs:

First Bug:

In first always block not all regs are reseted

```
always @(posedge V_if.clk or negedge V_if.rst_n) begin
    if (!V_if.rst_n) begin
        wr_ptr <= 0;
        //reset is not complete
        V_if.wr_ack <=0;
        V_if.overflow <= 0;
    end
    else if (V_if.wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= V_if.data_in;
```

Second Bug:

Underflow was not handled sequentially

```
always @(posedge V_if.clk or negedge V_if.rst_n) begin
    if (!V_if.rst_n) begin
        rd_ptr <= 0;
        //underflow reset
        V_if.underflow <= 0;
    end
    else if (V_if.rd_en && count != 0) begin
        V_if.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
    //handling the underflow signal
    else begin
        if (V_if.empty & V_if.rd_en)
            V_if.underflow <= 1;
        else
            V_if.underflow <= 0;
    end
end
```

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

Third Bug:

Not handling the cases when rd_en and wr_en are 11 or 00

I used case to handle all cases

```
    else begin
        case ({V_if.wr_en,V_if.rd_en})
            2'b00 : begin
                V_if.data_out <= V_if.data_out;
            end
            2'b01 : begin
                if(!V_if.empty)
                    count <= count - 1;
            end
            2'b10 : begin
                if(!V_if.full)
                    count <= count + 1;
            end
            2'b11 : begin
                if(V_if.full)
                    count <= count - 1;
                else if(V_if.empty)
                    count <= count + 1;
                else
                    count <= count;
            end
        endcase
        // if   ( ({V_if.wr_en, V_if.rd_en} == 2'b10) && !V_if.full)
        //  count <= count + 1;
        // else if ( ({V_if.wr_en, V_if.rd_en} == 2'b01) && !V_if.empty)
        //  count <= count - 1;
    end
end
```

QuestaSim snippets:

```
# Element at index 1: 42017
# Simulation finished!
# Correct transactions: 32800
# Errors found: 0
# ** Note: $stop    : FIFO_monitor.sv(
#    Time: 65598 ns  Iteration: 1  Ins
```

Functional coverage:

Covergroups

Name	C	Coverage	Goal	% of Goal	Status	Included	Merge_
/FIFO_coverage_pkg/FIFO_coverage		100.00%					
TYPE fifo_cg		100.00%	100	100.00...		✓	
CVP fifo_cg::cp_we_en		100.00%	100	100.00...		✓	
CVP fifo_cg::cp_rd_en		100.00%	100	100.00...		✓	
CVP fifo_cg::cp_wr_ack_signal		100.00%	100	100.00...		✓	
CVP fifo_cg::cp_overflow_signal		100.00%	100	100.00...		✓	
CVP fifo_cg::cp_full_signal		100.00%	100	100.00...		✓	
CVP fifo_cg::cp_empty_signal		100.00%	100	100.00...		✓	
CVP fifo_cg::cp_almostfull_signal		100.00%	100	100.00...		✓	
CVP fifo_cg::cp_almostempty_signal		100.00%	100	100.00...		✓	
CVP fifo_cg::cp_underflow_signal		100.00%	100	100.00...		✓	
CROSS fifo_cg::wr_en_cross_rd_en		100.00%	100	100.00...		✓	
CROSS fifo_cg::wr_en_cross_wr_ack		100.00%	100	100.00...		✓	
CROSS fifo_cg::wr_en_cross_overflow		100.00%	100	100.00...		✓	
CROSS fifo_cg::wr_en_cross_full		100.00%	100	100.00...		✓	
CROSS fifo_cg::wr_en_cross_empty		100.00%	100	100.00...		✓	
CROSS fifo_cg::wr_en_cross_almostfull		100.00%	100	100.00...		✓	
CROSS fifo_cg::wr_en_cross_almostempty		100.00%	100	100.00...		✓	
CROSS fifo_cg::wr_en_cross_underflow		100.00%	100	100.00...		✓	
CROSS fifo_cg::rd_en_cross_wr_ack		100.00%	100	100.00...		✓	
CROSS fifo_cg::rd_en_cross_overflow		100.00%	100	100.00...		✓	
CROSS fifo_cg::rd_en_cross_full		100.00%	100	100.00...		✓	
CROSS fifo_cg::rd_en_cross_empty		100.00%	100	100.00...		✓	
CROSS fifo_cg::rd_en_cross_almostfull		100.00%	100	100.00...		✓	
CROSS fifo_cg::rd_en_cross_almostempty		100.00%	100	100.00...		✓	
CROSS fifo_cg::rd_en_cross_underflow		100.00%	100	100.00...		✓	

Assertion coverage:

Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active	Cumulati
a_reset	Immediate	SVA	on	0	1	-	
assert__almostempty_prop	Concurrent	SVA	on	0	1	-	
assert__almostfull_prop	Concurrent	SVA	on	0	1	-	
assert__empty_prop	Concurrent	SVA	on	0	1	-	
assert__full_prop	Concurrent	SVA	on	0	1	-	
assert__overflow_prop	Concurrent	SVA	on	0	1	-	
assert__reset_reg	Concurrent	SVA	on	0	1	-	
assert__underflow_prop	Concurrent	SVA	on	0	1	-	
assert__wr_ack_prop	Concurrent	SVA	on	0	1	-	
immed__46	Immediate	SVA	on	0	1	-	

Code coverage:

Statement:

Statements - by instance (/FIFO_top/DUT)

FIFO.sv

```
18 always @(posedge V_if.clk or negedge V_if.rst_n) begin
20 wr_ptr <= 0;
22 V_if.wr_ack <=0;
23 V_if.overflow <= 0;
26 mem[wr_ptr] <= V_if.data_in;
27 V_if.wr_ack <= 1;
28 wr_ptr <= wr_ptr + 1;
31 V_if.wr_ack <= 0;
33 V_if.overflow <= 1;
35 V_if.overflow <= 0;
39 always @(posedge V_if.clk or negedge V_if.rst_n) begin
41 rd_ptr <= 0;
42 V_if.underflow <= 0;
45 V_if.data_out <= mem[rd_ptr];
46 rd_ptr <= rd_ptr + 1;
51 V_if.underflow <= 1;
53 V_if.underflow <= 0;
57 always @(posedge V_if.clk or negedge V_if.rst_n) begin
59 count <= 0;
64 V_if.data_out <= V_if.data_out;
68 count <= count - 1;
72 count <= count + 1;
76 count <= count - 1;
78 count <= count + 1;
80 count <= count;
90 assign V_if.full = (count == FIFO_DEPTH)? 1 : 0;
91 assign V_if.empty = (count == 0)? 1 : 0;
93 assign V_if.almostfull = (count == FIFO_DEPTH-1)? 1 : 0;
94 assign V_if.almostempty = (count == 1)? 1 : 0;
98 always_comb begin
```

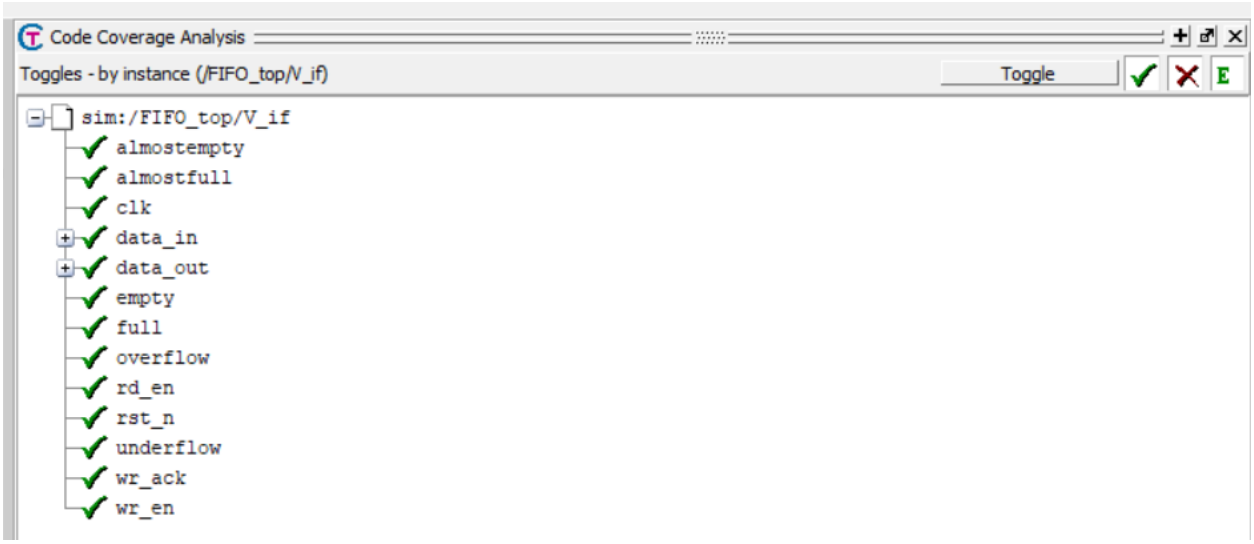
Branch:

Branches - by instance (/FIFO_top/DUT)

FIFO.sv

```
19 if (!V_if.rst_n) begin
25 else if (V_if.wr_en && count < FIFO_DEPTH) begin
30 else begin
32 if (V_if.full & V_if.wr_en)
34 else
40 if (!V_if.rst_n) begin
44 else if (V_if.rd_en && count != 0) begin
49 else begin
50 if (V_if.empty & V_if.rd_en)
52 else
58 if (!V_if.rst_n) begin
61 else begin
62 case ({V_if.wr_en,V_if.rd_en})
    62.1 case
63 2'b00 : begin
66 2'b01 : begin
67 if(!V_if.empty)
70 2'b10 : begin
71 if(!V_if.full)
74 2'b11 : begin
75 if(V_if.full)
77 else if(V_if.empty)
79 else
90 assign V_if.full = (count == FIFO_DEPTH)? 1 : 0;
91 assign V_if.empty = (count == 0)? 1 : 0;
93 assign V_if.almostfull = (count == FIFO_DEPTH-1)? 1 : 0;
94 assign V_if.almostempty = (count == 1)? 1 : 0;
99 if(!V_if.rst_n)
```

Toggle:



Verification Plan:

	A	B	C	D	E	F
1	**Label**	**Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**	
2	Reset	When the reset is asserted,all the internal signal registers should be low except the underflow to be 1	Directed at the start of the simulation		immidiate assertion to check reset	
3	FIFO_1	assert read and write enable to 1 after the reset then read	Directed	cover the wr_en with the rd_en in all cases	Concurrent assertion to check the empty flag & the underflow register & the almostempty flag	
4	FIFO_2	write 11 times in the FIFO to check the full flags & almost full & overflow	Directed	cover rd_en with all cases when fifo is empty or full ect.	Concurrent assertion to check the full flag & the overflow register & the almostfull flag & the wr_ack register	
5	FIFO_3	read 13 times in the FIFO to check the emty , almostempty and underflow flags	Directed	cover wr_en with all cases when fifo is empty or full ect.	Concurrent assertion to check the full flag & the overflow register & the wr_ack register	
6	FIFO_4	some directed assigns for the coverage	Directed		Concurrent assertion to check the empty flag & the underflow register & the almostempty flag	
7	FIFO_5	randomize for 2^15 times all the data baised on the req constraints	Randomized		Concurrent assertion to check all the flags & all the internal registers	
8						
9						