Synchronous FIFO

1. Verification plan

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	When reset is asserted, the internal pointers and counters should be zero	Directed at the start of the simulation	Cover reset flags: empty=1, full=0, almostfull=0, almostempty=0, wr_ack=0, overflow=0, underflow=0	Check the rst_n behavior through assertions and golden model
FIFO_2	When a write enable signal (wr_en) is active and the FIFO is not full, wr_ack should be asserted to confirm the write operation.	Directed during the simulation	Cover wr_ack bins; Cross wr_en, rd_en, wr_ack	Check the wr_ack through assertions
FIFO_3	If a write is attempted when the FIFO is full, overflow should be asserted.	Directed during the simulation	Cover overflow bins; Cross wr_en, rd_en, overflow (ignore invalid)	Check the overflow through assertions
FIFO_4	If a read is attempted when the FIFO is empty, underflow should be asserted.	Directed during the simulation	Cover underflow bins; Cross wr_en, rd_en, underflow (ignore invalid)	Check the underflow through assertions
FIFO_5	When the internal count is zero, the empty flag should be asserted.	Directed during the simulation	Cover empty bins; Cross wr_en, rd_en, empty (ignore invalid)	Check the empty flag through assertions
FIFO_6	When the internal count equals the FIFO depth, the full flag should be asserted.	Directed during the simulation	Cover full bins; Cross wr_en, rd_en, full (ignore invalid)	Check the full flag through assertions
FIFO_7	When the count reaches FIFO depth - 1, almostfull should be asserted.	Directed during the simulation	Cover almostfull bins; Cross wr_en, rd_en, almostfull	Check the almost full flag through assertions
FIFO_8	When the count equals 1, the almostempty signal should be asserted.	Directed during the simulation	Cover almostempty bins; Cross wr_en, rd_en, almostempty	Check the almost empty flag through assertions
FIFO_9	After writing or reading FIFO_DEPTH entries (0 to 7), the write or read pointer should eventually wrap around back to 0. Same applies for the counter (it should wrap from 8 to 0 with the reset).	Directed during the simulation	Cross full, empty, almostfull, almostempty with wr_en=1, rd_en=1 for wrap-around	Check the pointer wraparound through assertions and golden model
FIFO_10	Internal pointers cannot exceed the FIFO_DEPTH entries in any given time. Same applies for the counter.	Directed during the simulation	Cover boundary flags: full=1, almostfull=1, empty=1, almostempty=1	Check the pointer values through assertions
FIFO_11	when write_en is high , the data_in data should be stored in the FIFO	Directed during the simulation	Cover wr_en bins; Cross wr_en, rd_en with status flags for writes	Check the data_in through golden model
FIFO_12	when read_en is high , the data should be readed sequentially in data_out	Directed during the simulation	Cover rd_en bins; Cross wr_en, rd_en with status flags for reads	Check the data_out through golden model
FIFO_13	when asynchronous reset is asserted , all pointers should be zero and the FIFO should considered empty	Directed during the simulation	Cover post-async reset flags: empty=1, full=0, almostfull=0, almostempty=0	Check the rst_n behavior through assertions in the top module

2. Design edits

```
always @(posedge fifo_if.clk or negedge fifo_if.rst_n) begin
    if (!fifo_if.rst_n) begin
        wr_ptr <= 0;
        fifo_if.wr_ack <= 0;
        fifo_if.overflow <= 0; //improve
```

```
    else if (fifo_if.wr_en && count < fifo_if.FIFO_DEPTH) begin
        mem[wr_ptr] <= fifo_if.data_in;
        fifo_if.wr_ack <= 1;
        fifo_if.overflow <= 0; //improve
        if(wr_ptr == fifo_if.FIFO_DEPTH-1) //improve
            wr_ptr <= 0;
        else
            wr_ptr <= wr_ptr + 1;
    end
```

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

    //improve
    if (fifo_if.empty & fifo_if.rd_en && fifo_if.rst_n)
        fifo_if.underflow <= 1;
    else
        fifo_if.underflow <= 0;
end
```

```
156  always @(posedge fifo_if.clk or negedge fifo_if.rst_n) begin
157      if (!fifo_if.rst_n) begin
158          count <= 0;
159      end
160      else begin // improve
161          case ({fifo_if.wr_en, fifo_if.rd_en})
162              2'b10: if (!fifo_if.full) count <= count + 1;
163              2'b01: if (!fifo_if.empty) count <= count - 1;
164              2'b11: begin
165                  if (fifo_if.empty) count <= count + 1;
166                  else if (fifo_if.full) count <= count - 1;
167              end
168              default: count <= count;
169          endcase
170      end
171  end
172
173  assign fifo_if.full = (count == fifo_if.FIFO_DEPTH)? 1 : 0;
174  assign fifo_if.empty = (count == 0)? 1 : 0;
175  assign fifo_if.almostfull = (count == fifo_if.FIFO_DEPTH-1)? 1 : 0; //improve
176  assign fifo_if.almostempty = (count == 1)? 1 : 0;
177
178  endmodule
```

Design code :

FIFO.sv > FIFO

```
1   ////////////////////////////////////////////////////////////////////////////////
2   // Author: Kareem Waseem
3   // Course: Digital Verification using SV & UVM
4   //
5   // Description: FIFO Design
6   //
7   ////////////////////////////////////////////////////////////////////////////////
8   module FIFO(FIFO_if.DUT fifo_if);
9
10  localparam max_fifo_addr = $clog2(fifo_if.FIFO_DEPTH);
11
12  reg [fifo_if.FIFO_WIDTH-1:0] mem [fifo_if.FIFO_DEPTH-1:0];
13
14  reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
15  reg [max_fifo_addr:0] count;
16
17  `ifdef SIM
18  always_comb begin //FIFO_13
19      if(!fifo_if.rst_n) begin
20          count_rst : assert final(count == 0);
21          wr_ptr_rst : assert final(wr_ptr == 0);
22          rd_ptr_rst : assert final(rd_ptr == 0);
23      end
24      if(count ==1) begin
25          almostempty_assert : assert final(fifo_if.almostempty == 1'b1);
26      end
27      else if (count == 0) begin
28          empty_assert_comb : assert final(fifo_if.empty == 1'b1);
29      end
30      else if (count == fifo_if.FIFO_DEPTH) begin
31          full_assert_comb : assert final(fifo_if.full == 1'b1);
32      end
33      else if (count == fifo_if.FIFO_DEPTH-1) begin
34          almostfull_assert : assert final(fifo_if.almostfull == 1'b1);
35      end
36  end
37
38  property reset_behavior;
```

```
property reset_behavior;
endproperty

property write_ack;
  @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (fifo_if.wr_en && !fifo_if.full)
   |=> (fifo_if.wr_ack == 1);
endproperty

property overflow_detection;
  @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
   (fifo_if.wr_en && fifo_if.full) |=> (fifo_if.overflow == 1);
endproperty

property underflow_detection;
  @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
   (fifo_if.rd_en && fifo_if.empty) |=> (fifo_if.underflow == 1);
endproperty

property empty_flag;
  @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
   (count == 0) |-> (fifo_if.empty == 1);
endproperty

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

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

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

```
property write_ptr_wrap;
  @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
  (fifo_if.wr_en && !fifo_if.full && wr_ptr == fifo_if.FIFO_DEPTH-1) |-> (wr_ptr == 0);
endproperty

property read_ptr_wrap;
  @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
  (fifo_if.rd_en && !fifo_if.empty && rd_ptr == fifo_if.FIFO_DEPTH-1) |-> (rd_ptr == 0);
endproperty

property pointer_threshold;
  @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
   (wr_ptr < fifo_if.FIFO_DEPTH && rd_ptr < fifo_if.FIFO_DEPTH);
endproperty
property count_threshold;
  @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
   (count <= fifo_if.FIFO_DEPTH);
endproperty

FIFO_1 : assert property(reset_behavior);
FIFO_2 : assert property(write_ack);
FIFO_3 : assert property(overflow_detection);
FIFO_4 : assert property(underflow_detection);
FIFO_5 : assert property(empty_flag);
FIFO_6 : assert property(full_flag);
FIFO_7 : assert property(almostempty_flag);
FIFO_8 : assert property(almostfull_flag);
FIFO_9_0 : assert property(write_ptr_wrap);
FIFO_9_1 : assert property(read_ptr_wrap);
FIFO_10_0 : assert property(pointer_threshold);
FIFO_10_1 : assert property(count_threshold);

`endif

```

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

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

    //improve
    if (fifo_if.empty & fifo_if.rd_en && fifo_if.rst_n)
        fifo_if.underflow <= 1;
    else
        fifo_if.underflow <= 0;
end

always @(posedge fifo_if.clk or negedge fifo_if.rst_n) begin
    if (!fifo_if.rst_n) begin
        count <= 0;
    end
    else begin // improve
        case ({fifo_if.wr_en, fifo_if.rd_en})
            2'b10: if (!fifo_if.full) count <= count + 1;
            2'b01: if (!fifo_if.empty) count <= count - 1;
            2'b11: begin
                if (fifo_if.empty) count <= count + 1;
                else if (fifo_if.full) count <= count - 1;
            end
            default: count <= count;
        endcase
    end
end

assign fifo_if.full = (count == fifo_if.FIFO_DEPTH)? 1 : 0;
assign fifo_if.empty = (count == 0)? 1 : 0;
assign fifo_if.almostfull = (count == fifo_if.FIFO_DEPTH-1)? 1 : 0; //improve
assign fifo_if.almostempty = (count == 1)? 1 : 0;

endmodule
```

Interface:

```
interface FIFO_if #(parameter FIFO_DEPTH = 8, parameter FIFO_WIDTH = 16)(clk);
    input bit clk;
    logic rst_n;
    logic wr_en;
    logic rd_en;
    logic [FIFO_WIDTH-1:0] data_in;
    logic [FIFO_WIDTH-1:0] data_out;
    logic full;
    logic empty;
    logic almostfull;
    logic almostempty;
    logic wr_ack;
    logic overflow;
    logic underflow;

    modport DUT(
        input clk,
        input rst_n,
        input wr_en,
        input rd_en,
        input data_in,
        output data_out,
        output full,
        output empty,
        output almostfull,
        output almostempty,
        output wr_ack,
        output overflow,
        output underflow
    );
```

```
    modport TEST(
        input clk,
        output rst_n,
        output wr_en,
        output rd_en,
        output data_in,
        input data_out,
        input full,
        input empty,
        input almostfull,
        input almostempty,
        input wr_ack,
        input overflow,
        input underflow
    );
    modport mon(
        input clk,
        input rst_n,
        input wr_en,
        input rd_en,
        input data_in,
        input data_out,
        input full,
        input empty,
        input almostfull,
        input almostempty,
        input wr_ack,
        input overflow,
        input underflow
    );

endinterface
```

Shared package:

shared_pkg.sv > {} shared_pkg

```
package shared_pkg;

logic error_cnt;
logic correct_cnt;
bit test_finished;
event sample_event;

endpackage : shared_pkg
```

Transaction package:

FIFO_transaction.sv > {} FIFO_transaction_pkg > FIFO_transaction

```
package FIFO_transaction_pkg;

class FIFO_transaction;
  parameter FIFO_WIDTH = 8;
  bit clk;
  rand logic rst_n;
  rand logic wr_en;
  rand logic rd_en;
  rand logic [FIFO_WIDTH-1:0] data_in;
  logic [FIFO_WIDTH-1:0] data_out;
  logic full;
  logic empty;
  logic almostfull;
  logic almostempty;
  logic wr_ack;
  logic overflow;
  logic underflow;

  integer RD_EN_ON_DIST;
  integer WR_EN_ON_DIST;

  function new(integer RD_EN_ON_DIST = 30, integer WR_EN_ON_DIST = 70);
    this.RD_EN_ON_DIST = RD_EN_ON_DIST;
    this.WR_EN_ON_DIST = WR_EN_ON_DIST;
  endfunction : new

  constraint reset_c { rst_n dist{0:=10, 1:=90};}
  constraint wr_en_c { wr_en dist{0:=100-WR_EN_ON_DIST, 1:=WR_EN_ON_DIST};}
  constraint rd_en_c { rd_en dist{0:=100-RD_EN_ON_DIST, 1:=RD_EN_ON_DIST};}

endclass : FIFO_transaction
endpackage : FIFO_transaction_pkg
```

Coverage package:

```
package FIFO_cov_pkg;

import FIFO_transaction_pkg::*;

class FIFO_coverage;
  FIFO_transaction F_cvg_txn;

  covergroup FIFO_cvg_cg;
    write_en : coverpoint F_cvg_txn.wr_en{
        bins WR_EN_0 = {0};
        bins WR_EN_1 = {1};
    }
    read_en : coverpoint F_cvg_txn.rd_en{
        bins RD_EN_0 = {0};
        bins RD_EN_1 = {1};
    }
    full : coverpoint F_cvg_txn.full{
        bins FULL_0 = {0};
        bins FULL_1 = {1};
    }
    empty : coverpoint F_cvg_txn.empty{
        bins EMPTY_0 = {0};
        bins EMPTY_1 = {1};
    }
    almostfull : coverpoint F_cvg_txn.almostfull{
        bins ALMOSTFULL_0 = {0};
        bins ALMOSTFULL_1 = {1};
    }
    almostempty : coverpoint F_cvg_txn.almostempty{
        bins ALMOSTEMPTY_0 = {0};
        bins ALMOSTEMPTY_1 = {1};
    }
    overflow : coverpoint F_cvg_txn.overflow{
        bins OVERFLOW_0 = {0};
        bins OVERFLOW_1 = {1};
    }
```

```
        bins OVERFLOW_1 = {1};
    }
    underflow : coverpoint F_cvg_txn.underflow{
        bins UNDERFLOW_0 = {0};
        bins UNDERFLOW_1 = {1};
    }
    wr_ack : coverpoint F_cvg_txn.wr_ack{
        bins WR_ACK_0 = {0};
        bins WR_ACK_1 = {1};
    }
    cross_1 : cross write_en, read_en , full{
        ignore_bins cross_11 = binsof(write_en.WR_EN_1) && binsof(full.FULL_1);
    }
    cross_2 : cross write_en, read_en , empty{
        ignore_bins cross_22 = binsof(read_en.RD_EN_1) && binsof(empty.EMPTY_1);
    }

    cross_3 : cross write_en, read_en , almostfull;
    cross_4 : cross write_en, read_en , almostempty;

    cross_5 : cross write_en, read_en , overflow{
        ignore_bins cross_55 = binsof(write_en.WR_EN_0) && binsof(overflow.OVERFLOW_1);
    }

    cross_6 : cross write_en, read_en , underflow{
        ignore_bins cross_66 = binsof(read_en.RD_EN_0) && binsof(underflow.UNDERFLOW_1);
    }

    cross_7 : cross write_en, read_en , wr_ack{
        ignore_bins cross_77 = binsof(write_en.WR_EN_0) && binsof(wr_ack.WR_ACK_1);
    }
  endgroup : FIFO_cvg_cg
```

```
  function new();
    FIFO_cvg_cg = new();
  endfunction : new

  function void sample_data(FIFO_transaction F_txn);
        F_cvg_txn = F_txn;
        FIFO_cvg_cg.sample();
  endfunction : sample_data

endclass : FIFO_coverage
endpackage : FIFO_cov_pkg
```

Score board:

```
package FIFO_sb_pkg;

import FIFO_transaction_pkg::*;
import shared_pkg::*;

parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;

class FIFO_sb;
  logic [FIFO_WIDTH-1:0] data_out_ref;
  logic [FIFO_WIDTH-1:0] fifo_mem[$];
  reg [3:0] size;
  int count = 0 ;

  function new();
    error_cnt = 0;
    correct_cnt = 0;
  endfunction : new

  //FIFO_12
  task check_data(FIFO_transaction F_txn);
    reference_model(F_txn);
    if (F_txn.data_out !== data_out_ref)
      begin
        $display("Mismatch Detected:");
        $display(" DUT: data_out=%0h ", F_txn.data_out );
        $display(" REF: data_out=%0h ", data_out_ref);
        error_cnt++;
      end
    else
        correct_cnt++;

  endtask : check_data
```

```
  task reference_model(FIFO_transaction F_txn);
      size = fifo_mem.size();
      if(!F_txn.rst_n) begin
        data_out_ref = 0;
        fifo_mem.delete();
        size = fifo_mem.size();
      end
      else begin
        size = fifo_mem.size();
        case ({F_txn.wr_en,F_txn.rd_en})
        2'b10: begin
           if (size != 8) begin
            fifo_mem.push_back(F_txn.data_in);
            data_out_ref = data_out_ref;
            // data_out_ref remains unchanged
            end
        end
        // Read only
        2'b01: begin
          if (size != 0) begin
            data_out_ref = fifo_mem.pop_front();
          end
        end
        // Read + Write simultaneously
        2'b11: begin
          if (size == 0) begin
            fifo_mem.push_back(F_txn.data_in);
          end
          else if (size == 8) begin
            data_out_ref = fifo_mem.pop_front();
          end
          else begin
            data_out_ref = fifo_mem.pop_front();
            fifo_mem.push_back(F_txn.data_in);
          end
```

Monitor:

monitor.sv > monitor

```
import FIFO_transaction_pkg::*;
import FIFO_sb_pkg::*;
import FIFO_cov_pkg::*;
import shared_pkg::*;

module monitor(FIFO_if.mon fifo_if);
FIFO_transaction fifo_txn;
FIFO_sb fifo_sb;
FIFO_coverage fifo_cvg;
```

```
initial
  begin
    fifo_txn = new(65,65);
    fifo_sb = new();
    fifo_cvg = new();
    forever begin
        // Wait for event triggered by testbench after driving inputs
        wait(sample_event.triggered);
        @(negedge fifo_if.clk);
        fifo_txn.rst_n = fifo_if.rst_n;
        fifo_txn.wr_en = fifo_if.wr_en;
        fifo_txn.rd_en = fifo_if.rd_en;
        fifo_txn.data_in = fifo_if.data_in;
        fifo_txn.data_out = fifo_if.data_out;
        fifo_txn.full = fifo_if.full;
        fifo_txn.empty = fifo_if.empty;
        fifo_txn.almostfull = fifo_if.almostfull;
        fifo_txn.almostempty = fifo_if.almostempty;
        fifo_txn.wr_ack = fifo_if.wr_ack;
        fifo_txn.overflow = fifo_if.overflow;
        fifo_txn.underflow = fifo_if.underflow;
        fork
            fifo_cvg.sample_data(fifo_txn);
            fifo_sb.check_data(fifo_txn);
        join
        if(test_finished) begin
            $display("Total Errors = %0d, Total Correct = %0d", error_cnt, correct_cnt);
            $stop;
        end
    end
  end

endmodule
```

Tb:

FIFO_tb.sv > FIFO_tb

```
import FIFO_transaction_pkg::*;
import shared_pkg::*;

module FIFO_tb(FIFO_if.TEST fifo_if);
FIFO_transaction fifo_txn;

initial
  begin
    fifo_txn = new(75,75);
    fifo_if.wr_en = 0;
    fifo_if.rd_en = 0;
    fifo_if.data_in = 16'h0000;
    assert_reset();
    for(int i=0; i<25000; i++) begin
        assert(fifo_txn.randomize());
        fifo_if.rst_n = fifo_txn.rst_n;
        fifo_if.wr_en = fifo_txn.wr_en;
        fifo_if.rd_en = fifo_txn.rd_en;
        fifo_if.data_in = fifo_txn.data_in;
        @(negedge fifo_if.clk);

        -> sample_event;
    end
    test_finished = 1;
  end

  task assert_reset();
    fifo_if.rst_n = 0;
    @(negedge fifo_if.clk);
    fifo_if.rst_n = 1;
    @(negedge fifo_if.clk);
  endtask : assert_reset



endmodule : FIFO_tb
```

Top module:

top.sv > top

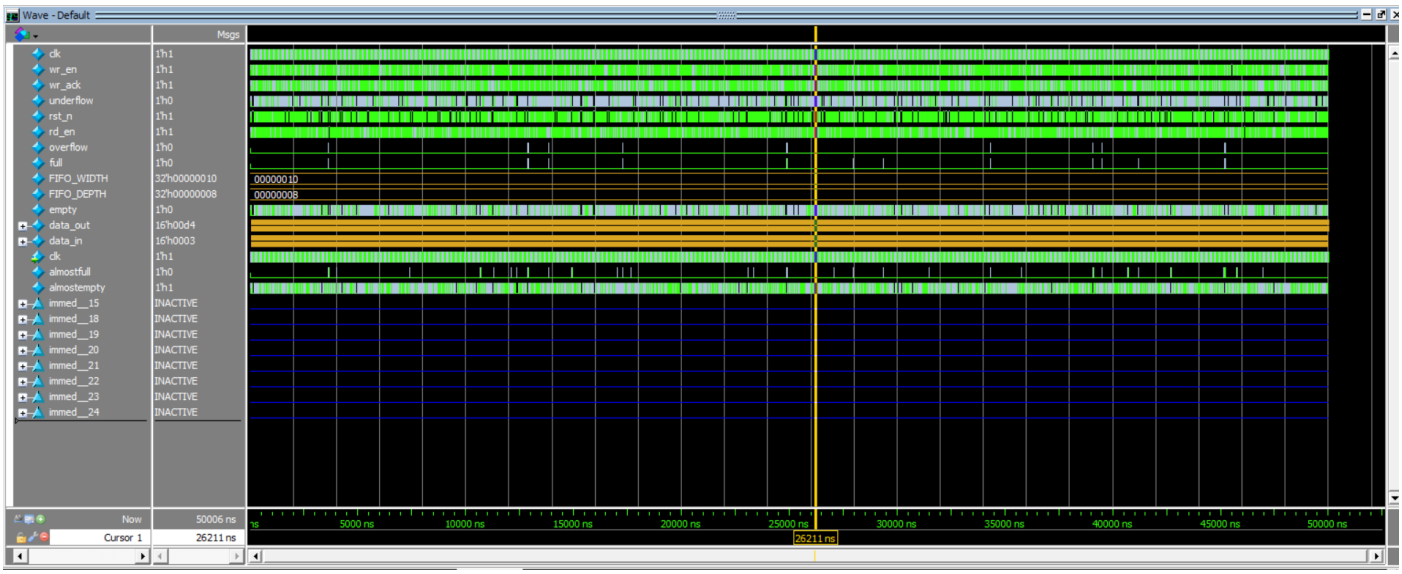
```
module top();

  bit clk;

initial
  begin
    clk = 0;
    forever #1 clk = ~clk;
  end

  FIFO_if fifo_if(clk);
  FIFO DUT(fifo_if);
  FIFO_tb tb(fifo_if);
  monitor mon(fifo_if);

  always_comb begin
    if(!fifo_if.rst_n) begin
        assert final (fifo_if.wr_ack == 0);
        assert final (fifo_if.overflow == 0);
        assert final (fifo_if.underflow == 0);
        assert final (fifo_if.full == 0);
        assert final (fifo_if.empty == 1);
        assert final (fifo_if.almostfull == 0);
        assert final (fifo_if.almostempty == 0);
    end
  end

endmodule
```

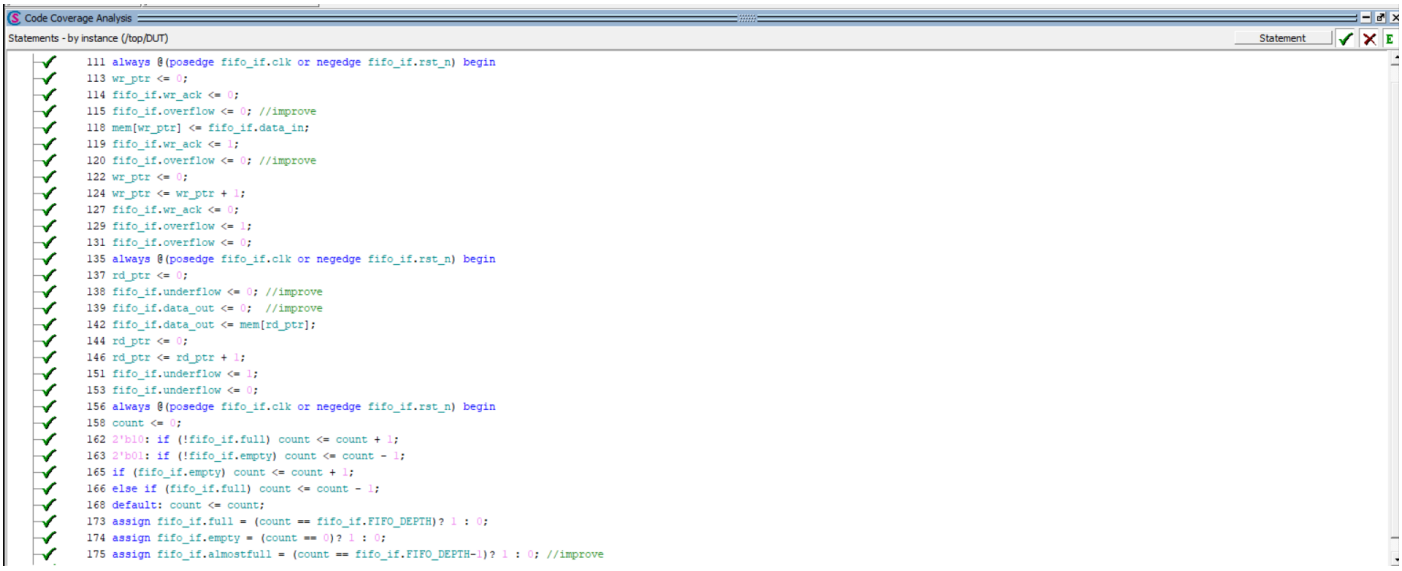
Do file:

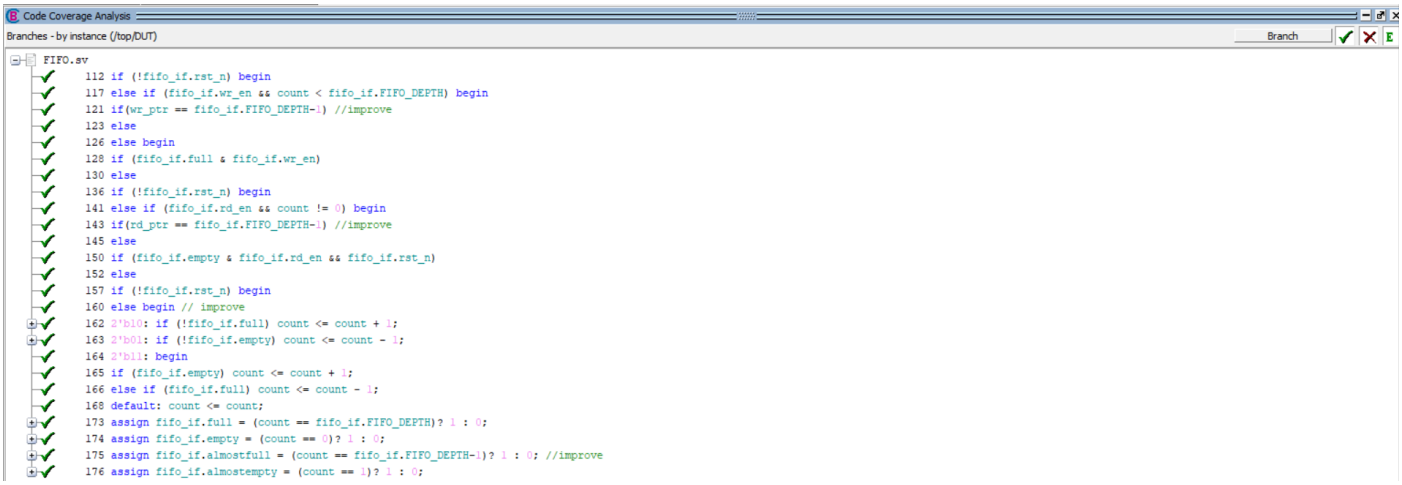
run.do

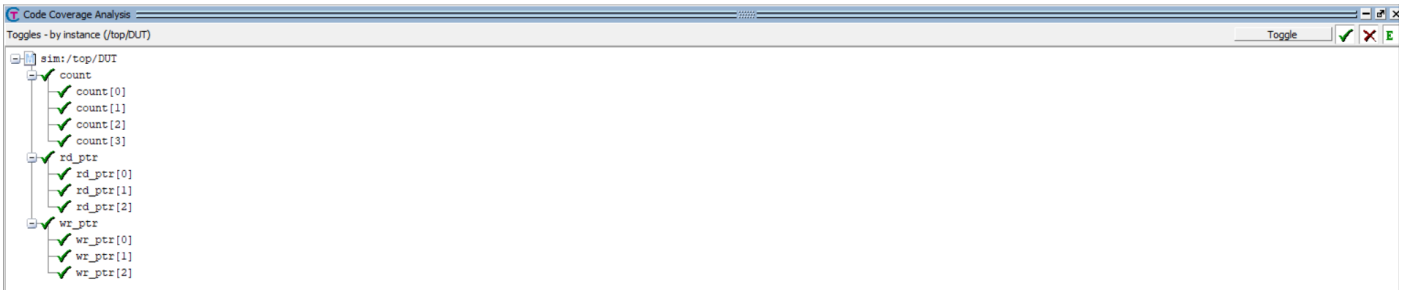
```
vlib work
vlog shared_pkg.sv FIFO_transaction.sv FIFO_cov.sv FIFO_sb.sv FIFO.sv FIFO_if.sv FIFO_tb.sv monitor.sv top.sv +cover -covercells
vsim -voptargs=+acc work.top -cover
add wave *
add wave -position insertpoint  \
sim:/top/fifo_if/wr_en \
sim:/top/fifo_if/wr_ack \
sim:/top/fifo_if/underflow \
sim:/top/fifo_if/rst_n \
sim:/top/fifo_if/rd_en \
sim:/top/fifo_if/overflow \
sim:/top/fifo_if/full \
sim:/top/fifo_if/FIFO_WIDTH \
sim:/top/fifo_if/FIFO_DEPTH \
sim:/top/fifo_if/empty \
sim:/top/fifo_if/data_out \
sim:/top/fifo_if/data_in \
sim:/top/fifo_if/clk \
sim:/top/fifo_if/almostfull \
sim:/top/fifo_if/almostempty
coverage save top.ucdb -onexit -du work.ALSU
run -all

```

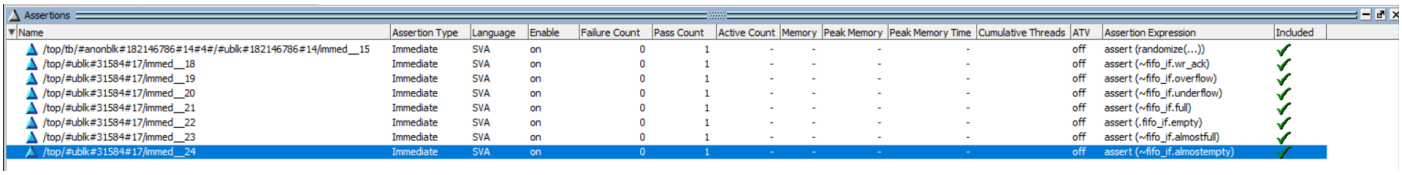
Questa snippet:











Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
/top/tb/#anonblk#182146786#14#4#/#ublk#182146786#14/immed__15	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert (randomize(...))	✓
/top/#ublk#31584#17/immed__18	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert (~fifo_if.wr_ack)	✓
/top/#ublk#31584#17/immed__19	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert (~fifo_if.overflow)	✓
/top/#ublk#31584#17/immed__20	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert (~fifo_if.underflow)	✓
/top/#ublk#31584#17/immed__21	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert (~fifo_if.full)	✓
/top/#ublk#31584#17/immed__22	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert (.fifo_if.empty)	✓
/top/#ublk#31584#17/immed__23	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert (~fifo_if.almostfull)	✓
/top/#ublk#31584#17/immed__24	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert (~fifo_if.almostempty)	✓

Transcript

```
coverage report -assert -detail /.
# Coverage Report by instance with details
#
# ================================================================================
# === Instance: /top/tb
# === Design Unit: work.FIFO_tb
# ================================================================================
#
# Assertion Coverage:
#     Assertions                     1         1         0   100.00%
# --------------------------------------------------------------------------------
# Name                   File(Line)                    Failure      Pass
#                                                      Count        Count
# --------------------------------------------------------------------------------
# /top/tb/#anonblk#182146786#144#/#ublk#182146786#14/immed__15
#                        FIFO_tb.sv(15)                      0          1
#
# ================================================================================
# === Instance: /top
# === Design Unit: work.top
# ================================================================================
#
# Assertion Coverage:
#     Assertions                     7         7         0   100.00%
# --------------------------------------------------------------------------------
# Name                   File(Line)                    Failure      Pass
#                                                      Count        Count
# --------------------------------------------------------------------------------
# /top/#ublk#31584#17/immed__24
#                        top.sv(24)                          0          1
# /top/#ublk#31584#17/immed__23
#                        top.sv(23)                          0          1
# /top/#ublk#31584#17/immed__22
#                        top.sv(22)                          0          1
# /top/#ublk#31584#17/immed__21
#                        top.sv(21)                          0          1
# /top/#ublk#31584#17/immed__20
#                        top.sv(20)                          0          1
# /top/#ublk#31584#17/immed__19
```

```
#
#
# ASSERTION RESULTS:
# --------------------------------------------------------------------------------
# Name                   File(Line)                    Failure      Pass
#                                                      Count        Count
# --------------------------------------------------------------------------------
# /top/#ublk#31584#17/immed__24
#                        top.sv(24)                          0          1
# /top/#ublk#31584#17/immed__23
#                        top.sv(23)                          0          1
# /top/#ublk#31584#17/immed__22
#                        top.sv(22)                          0          1
# /top/#ublk#31584#17/immed__21
#                        top.sv(21)                          0          1
# /top/#ublk#31584#17/immed__20
#                        top.sv(20)                          0          1
# /top/#ublk#31584#17/immed__19
#                        top.sv(19)                          0          1
# /top/#ublk#31584#17/immed__18
#                        top.sv(18)                          0          1
# /top/tb/#anonblk#182146786#144#/#ublk#182146786#14/immed__15
#                        FIFO_tb.sv(15)                      0          1
#
# Total Coverage By Instance (filtered view): 100.00%
```

Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/FIFO_cov_pkg/FIFO_coverage		100.00%							
TYPE FIFO_cvg_cg		100.00%	100	100.00...		✓	auto(1)		
CVP FIFO_cvg_cg::write_en		100.00%	100	100.00...		✓			
CVP FIFO_cvg_cg::read_en		100.00%	100	100.00...		✓			
CVP FIFO_cvg_cg::full		100.00%	100	100.00...		✓			
CVP FIFO_cvg_cg::empty		100.00%	100	100.00...		✓			
CVP FIFO_cvg_cg::almostfull		100.00%	100	100.00...		✓			
CVP FIFO_cvg_cg::almostempty		100.00%	100	100.00...		✓			
CVP FIFO_cvg_cg::overflow		100.00%	100	100.00...		✓			
CVP FIFO_cvg_cg::underflow		100.00%	100	100.00...		✓			
CVP FIFO_cvg_cg::wr_ack		100.00%	100	100.00...		✓			
CROSS FIFO_cvg_cg::cross_2		100.00%	100	100.00...		✓			
bin <WR_EN_1,RD_EN_1,EMPTY_0>		12650	1	100.00...		✓			
bin <WR_EN_0,RD_EN_1,EMPTY_0>		1515	1	100.00...		✓			
bin <WR_EN_1,RD_EN_0,EMPTY_1>		461	1	100.00...		✓			
bin <WR_EN_1,RD_EN_0,EMPTY_0>		4244	1	100.00...		✓			
bin <WR_EN_0,RD_EN_0,EMPTY_1>		474	1	100.00...		✓			
bin <WR_EN_0,RD_EN_0,EMPTY_0>		1099	1	100.00...		✓			
ignore_bin cross_22		4557	-	-		✓			
CROSS FIFO_cvg_cg::cross_3		100.00%	100	100.00...		✓			
bin <WR_EN_1,RD_EN_1,ALMOSTFU...		90	1	100.00...		✓			
bin <WR_EN_0,RD_EN_1,ALMOSTFU...		3	1	100.00...		✓			
bin <WR_EN_1,RD_EN_0,ALMOSTFU...		41	1	100.00...		✓			
bin <WR_EN_0,RD_EN_0,ALMOSTFU...		11	1	100.00...		✓			
bin <WR_EN_1,RD_EN_1,ALMOSTFU...		13984	1	100.00...		✓			
bin <WR_EN_0,RD_EN_1,ALMOSTFU...		4645	1	100.00...		✓			
bin <WR_EN_1,RD_EN_0,ALMOSTFU...		4664	1	100.00...		✓			
bin <WR_EN_0,RD_EN_0,ALMOSTFU...		1562	1	100.00...		✓			
CROSS FIFO_cvg_cg::cross_4		100.00%	100	100.00...		✓			
bin <WR_EN_1,RD_EN_1,ALMOSTE...		7973	1	100.00...		✓			
bin <WR_EN_0,RD_EN_1,ALMOSTE...		779	1	100.00...		✓			
bin <WR_EN_1,RD_EN_0,ALMOSTE...		908	1	100.00...		✓			
bin <WR_EN_0,RD_EN_0,ALMOSTE...		565	1	100.00...		✓			
bin <WR_EN_1,RD_EN_1,ALMOSTE...		6101	1	100.00...		✓			
bin <WR_EN_0,RD_EN_1,ALMOSTE...		3869	1	100.00...		✓			
CROSS FIFO_cvg_cg::cross_5		100.00%	100	100.00...		✓			
bin <WR_EN_1,RD_EN_1,OVERFLO...		18	1	100.00...		✓			
bin <WR_EN_1,RD_EN_0,OVERFLO...		4	1	100.00...		✓			
bin <WR_EN_1,RD_EN_1,OVERFLO...		14056	1	100.00...		✓			
bin <WR_EN_0,RD_EN_1,OVERFLO...		4648	1	100.00...		✓			
bin <WR_EN_1,RD_EN_0,OVERFLO...		4701	1	100.00...		✓			
bin <WR_EN_0,RD_EN_0,OVERFLO...		1573	1	100.00...		✓			
ignore_bin cross_55		0	-	-		✓			
CROSS FIFO_cvg_cg::cross_6		100.00%	100	100.00...		✓			
bin <WR_EN_1,RD_EN_1,UNDERFLO...		2766	1	100.00...		✓			
bin <WR_EN_1,RD_EN_1,UNDERFLO...		11308	1	100.00...		✓			
bin <WR_EN_0,RD_EN_1,UNDERFLO...		953	1	100.00...		✓			
bin <WR_EN_0,RD_EN_1,UNDERFLO...		3695	1	100.00...		✓			
bin <WR_EN_1,RD_EN_0,UNDERFLO...		4705	1	100.00...		✓			
bin <WR_EN_0,RD_EN_0,UNDERFLO...		1573	1	100.00...		✓			
ignore_bin cross_66		0	-	-		✓			
CROSS FIFO_cvg_cg::cross_7		100.00%	100	100.00...		✓			
bin <WR_EN_1,RD_EN_1,WR_ACK_...		12632	1	100.00...		✓			
bin <WR_EN_1,RD_EN_0,WR_ACK_...		4240	1	100.00...		✓			
bin <WR_EN_1,RD_EN_1,WR_ACK_...		1442	1	100.00...		✓			
bin <WR_EN_0,RD_EN_1,WR_ACK_...		4648	1	100.00...		✓			
bin <WR_EN_1,RD_EN_0,WR_ACK_...		465	1	100.00...		✓			
bin <WR_EN_0,RD_EN_0,WR_ACK_...		1573	1	100.00...		✓			
ignore_bin cross_77		0	-	-		✓			