FIFO

Synchronous FIFO Verification project

Prepared by
ABDELRAHMAN MAHMOUD

Contents

DESIGN SPECIFICATION:

Parameters

- FIFO_WIDTH: DATA in/out and memory word width (default: 16)
- FIFO_DEPTH: Memory depth (default: 8)

Ports

Port	Direction	Function
data_in	Input	Write Data: The input data bus used when writing the FIFO.
wr_en		Write Enable: If the FIFO is not full, asserting this signal causes data (on data_in) to be written into the FIFO
rd_en		Read Enable: If the FIFO is not empty, asserting this signal causes data (on data_out) to be read from the FIFO
clk		Clock signal
rst_n		Active low asynchronous reset
data_out	Output	Read Data: The sequential output data bus used when reading from the FIFO.
full		Full Flag: When asserted, this combinational output signal indicates that the FIFO is full. Write requests are ignored when the FIFO is full, initiating a write when the FIFO is full is not destructive to the contents of the FIFO.
almostfull		Almost Full: When asserted, this combinational output signal indicates that only one more write can be performed before the FIFO is full.
empty		Empty Flag: When asserted, this combinational output signal indicates that the FIFO is empty. Read requests are ignored when the FIFO is empty, initiating a read while empty is not destructive to the FIFO.
almostempty		Almost Empty: When asserted, this output combinational signal indicates that only one more read can be performed before the FIFO goes to empty.
overflow		Overflow: This sequential output signal indicates that a write request (wr_en) was rejected because the FIFO is full. Overflowing the FIFO is not destructive to the contents of the FIFO.
underflow		Underflow: This sequential output signal Indicates that the read request (rd_en) was rejected because the FIFO is empty. Under flowing the FIFO is not destructive to the FIFO.
wr_ack		Write Acknowledge: This sequential output signal indicates that a write request (wr_en) has succeeded.

DESIGN BUGS:

BUG #1: Missing reset for wr_ack & overflow

- **Original code:**

```
always @(posedge intf.clk or negedge intf.rst_n) begin
        if (!intf.rst_n) begin
            wr_ptr <= 0;
            intf.wr_ack <= 0; // Fix 1: Explicit reset for wr_ack
            intf.overflow <= 0; // Fix 1: Explicit reset for overflow
        end
```

- **Fixed code:**

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        wr_ptr <= 0;
    end
```

BUG #2: Missing simultaneous Read/Write handling

- **Original code:**

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rd_ptr <= 0;
    end
```

- **Fixed code:**

```
always @(posedge intf.clk or negedge intf.rst_n) begin
        if (!intf.rst_n) begin
            rd_ptr <= 0;
            intf.data_out <= 0;
            intf.underflow <= 0;
        end
```

BUG #3: incorrect almostfull assignment

- **Original code:**

```
1   assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
```

- **Fixed code:**

```
1   assign intf.almostfull   = (count == FIFO_DEPTH - 1) ? 1 : 0 ;
```

BUG #4: incorrect underflow assignment

- **Original code:**

```
assign underflow = (empty && rd_en)? 1 : 0;
```

- **Fixed code:**

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

- **Underflow was implemented as combinational signal while it should be registered to match error flags.**

BUG #5: Missing simultaneous Read/Write handling

- **Original code:**

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

- **Fixed code:**

```
always @(posedge intf.clk or negedge intf.rst_n) begin
        if (!intf.rst_n) begin
            count <= 0;
        end
        else begin
            if  ( ({intf.wr_en, intf.rd_en} == WR_ONLY && !intf.full) ||
                  ({intf.wr_en, intf.rd_en} == TWO_EN && intf.empty) )
                count <= count + 1;

            else if ( ({intf.wr_en, intf.rd_en} == RD_ONLY && !intf.empty) ||
                      ({intf.wr_en, intf.rd_en} == TWO_EN && intf.full) )
                count <= count - 1;
        end
    end
```

ORIGINAL DESIGN CODE:

```
module FIFO(data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull, almostempty, wr_ack, overflow, underflow, data_out);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
input [FIFO_WIDTH-1:0] data_in;
input clk, rst_n, wr_en, rd_en;
output reg [FIFO_WIDTH-1:0] data_out;
output reg wr_ack, overflow;
output full, empty, almostfull, almostempty, underflow;

localparam max_fifo_addr = $clog2(FIFO_DEPTH);

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

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

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

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

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

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

endmodule
```

MODIFIED DESIGN CODE WITH SVA:

```
module FIFO #(parameter FIFO_DEPTH = 8,parameter FIFO_WIDTH = 16)(FIFO_if.DUT intf);

    localparam max_fifo_addr = $clog2(FIFO_DEPTH);

    reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];
    reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
    reg [max_fifo_addr:0] count;

    localparam TWO_EN = 2'b11;
    localparam WR_ONLY = 2'b10;
    localparam RD_ONLY = 2'b01;

    always @(posedge intf.clk or negedge intf.rst_n) begin
        if (!intf.rst_n) begin
            wr_ptr <= 0;
            intf.wr_ack <= 0; // Fix 1: Explicit reset for wr_ack
            intf.overflow <= 0; // Fix 1: Explicit reset for overflow
        end
        else if (intf.wr_en && count < FIFO_DEPTH) begin
            mem[wr_ptr] <= intf.data_in;
            intf.wr_ack <= 1;
            wr_ptr <= (wr_ptr == FIFO_DEPTH-1) ? 0 : wr_ptr + 1;
            intf.overflow <= 0;
        end
        else begin
            intf.wr_ack <= 0;
            if (intf.full & intf.wr_en)
                intf.overflow <= 1;
            else
                intf.overflow <= 0;
        end
    end

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

    always @(posedge intf.clk or negedge intf.rst_n) begin
        if (!intf.rst_n) begin
            count <= 0;
        end
        else begin
            if  ( ({intf.wr_en, intf.rd_en} == WR_ONLY && !intf.full) ||
                  ({intf.wr_en, intf.rd_en} == TWO_EN && intf.empty) )
                count <= count + 1;

            else if ( ({intf.wr_en, intf.rd_en} == RD_ONLY && !intf.empty) ||
                      ({intf.wr_en, intf.rd_en} == TWO_EN && intf.full) )
                count <= count - 1;
        end
    end

    assign intf.full        = (count == FIFO_DEPTH) ? 1 : 0 ;
    assign intf.empty       = (count == 0) ? 1 : 0 ;
    assign intf.almostfull  = (count == FIFO_DEPTH - 1) ? 1 : 0 ; // Fix 3: Almostfull at DEPTH - 1
    assign intf.almostempty = (count == 1) ? 1 : 0 ;

```

```
//FIFO_1
////////////////////////////////
// b. Write Acknowledge
property write_ack;
    @(posedge intf.clk) disable iff(!intf.rst_n)
        (intf.wr_en && !intf.full) |=> (intf.wr_ack);
endproperty
assert property(write_ack)
else $error("Assertion FAILED: wr_ack not set correctly after write");

// c. Overflow Detection
property overflow_detection;
    @(posedge intf.clk) disable iff(!intf.rst_n)
        (intf.wr_en && intf.full) |=> (intf.overflow);
endproperty
assert property(overflow_detection)
else $error("Assertion FAILED: Overflow flag not set correctly when writing to full FIFO");

// d. Underflow Detection
property underflow_detection;
    @(posedge intf.clk) disable iff(!intf.rst_n)
        (intf.rd_en && intf.empty) |=> (intf.underflow);
endproperty
assert property(underflow_detection)
else $error("Assertion FAILED: Underflow flag not set correctly when reading from empty FIFO");

// e. Empty Flag Assertion
property Empty_Flag_Assertion;
    @(posedge intf.clk) disable iff(!intf.rst_n)
        (count==0) |-> (intf.empty);
endproperty
assert property(Empty_Flag_Assertion)
else $error("Assertion FAILED: empty flag mismatch with count==0");

// f. Full Flag Assertion
property Full_Flag_Assertion;
    @(posedge intf.clk) disable iff(!intf.rst_n)
        (count==FIFO_DEPTH) |-> (intf.full);
endproperty
assert property(Full_Flag_Assertion)
else $error("Assertion FAILED: full flag mismatch with count==FIFO_DEPTH");

```

```
    // g. Almost Full Condition
      property Almost_Full_Condition;
          @(posedge intf.clk) disable iff(!intf.rst_n)
              (count==FIFO_DEPTH-1) |-> (intf.almostfull);
      endproperty
      assert property(Almost_Full_Condition)
      else $error("Assertion FAILED: almostfull flag mismatch with count==FIFO_DEPTH-2");

      // h. Almost Empty Condition
      property Almost_empty_Condition;
          @(posedge intf.clk) disable iff(!intf.rst_n)
              (count==1) |-> (intf.almostempty);
      endproperty
      assert property(Almost_empty_Condition)
      else $error("Assertion FAILED: almostempty flag mismatch with count==1");

      // i. Write Pointer Wraparound
      property write_pointer_wraparound;
          @(posedge intf.clk) disable iff(!intf.rst_n)
              (wr_ptr==FIFO_DEPTH-1 && intf.wr_en && !intf.full) |=> (wr_ptr==0);
      endproperty
      assert property(write_pointer_wraparound)
      else $error("Assertion FAILED: wr_ptr did not wrap around to 0");

      // j. Read Pointer Wraparound
      property read_pointer_wraparound;
          @(posedge intf.clk) disable iff(!intf.rst_n)
              (rd_ptr==FIFO_DEPTH-1 && intf.rd_en && !intf.empty) |=> (rd_ptr==0);
      endproperty
      assert property(read_pointer_wraparound)
      else $error("Assertion FAILED: rd_ptr did not wrap around to 0");

      // k. Write Pointer Threshold
      property write_pointer_threshold;
          @(posedge intf.clk) wr_ptr inside {[0:FIFO_DEPTH-1]};
      endproperty
      assert property(write_pointer_threshold)
      else $error("Assertion FAILED: wr_ptr exceeded valid FIFO range");

      // l. Read Pointer Threshold
      property read_pointer_threshold;
          @(posedge intf.clk) rd_ptr inside {[0:FIFO_DEPTH-1]};
      endproperty
      assert property(read_pointer_threshold)
      else $error("Assertion FAILED: rd_ptr exceeded valid FIFO range");

      // m. Count Threshold
      property count_threshold;
          @(posedge intf.clk) count inside {[0:FIFO_DEPTH]};
      endproperty
      assert property(count_threshold)
      else $error("Assertion FAILED: count exceeded FIFO_DEPTH");

      // ---- coverage points ----
      cover property(after_reset);
      cover property(write_ack);
      cover property(overflow_detection);
      cover property(underflow_detection);
      cover property(Empty_Flag_Assertion);
      cover property(Full_Flag_Assertion);
      cover property(Almost_Full_Condition);
      cover property(Almost_empty_Condition);
      cover property(write_pointer_wraparound);
      cover property(read_pointer_wraparound);
      cover property(write_pointer_threshold);
      cover property(read_pointer_threshold);
      cover property(count_threshold);
  `endif
  endmodule
```

Interface:

```
interface FIFO_if #(parameter FIFO_DEPTH = 8,parameter FIFO_WIDTH = 16)(input logic clk);
    logic [FIFO_WIDTH-1:0] data_in;
    logic  rst_n, wr_en, rd_en;

    // signals driven by the DUT must be var
    var logic [FIFO_WIDTH-1:0] data_out;
    var logic wr_ack, overflow ;
    var logic full, empty, almostfull, almostempty, underflow ;

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

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

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

endinterface

```

SHARED PACKAGE:

```
package shared_pkg;
    bit test_finished = 0;
    integer error_count = 0;
    integer correct_count = 0;
    // event to synchronize monitor after TB drives stimulus
    event sample_event;
endpackage

```

TOP MODULE:

```
module top();
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;

    logic clk;

    // clock
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    // interface instance (driven by top clock)
    FIFO_if #(FIFO_DEPTH,FIFO_WIDTH) intf (clk);

    // DUT (accepts interface modport)
    FIFO #(FIFO_DEPTH,FIFO_WIDTH) dut (intf.DUT);

    // instantiate TB to drive the interface (use TB modport)
    FIFO_tb tb_inst (intf.TB);

    // instantiate MONITOR
    monitor monitor_inst (intf.MONITOR);

    // checking the functionality of the asynchronous reset
    always_comb begin
        if (!intf.rst_n) begin
            async_reset_assertion: assert final(
                !intf.data_out  && !intf.full       && !intf.almostfull && !intf.underflow &&
                intf.empty      && !intf.almostempty && !intf.overflow   && !intf.wr_ack);
            async_reset_cover: cover final(!intf.data_out && !intf.full && !intf.almostfull && intf.empty && !intf.almostempty && !intf.overflow && !intf.underflow && !intf.wr_ack);
        end
    end

endmodule

```

MONITOR CLASS:

```
import transaction_pkg::*;
import coverage_pkg::*;
import scoreboard_pkg::*;
import shared_pkg::*;
module monitor #(parameter FIFO_DEPTH = 8,parameter FIFO_WIDTH = 16)(FIFO_if.MONITOR intf);

    FIFO_transaction trans = new();
    FIFO_coverage    cov = new();
    FIFO_scoreboard  score = new();

    initial begin
        forever begin
            wait(sample_event.triggered);
            @(negedge intf.clk)

            //assign interface data variables  to the data variables of the trans
            trans.data_in     = intf.data_in;
            trans.clk         = intf.clk;
            trans.rst_n       = intf.rst_n;
            trans.wr_en       = intf.wr_en;
            trans.rd_en       = intf.rd_en;
            trans.data_out    = intf.data_out;
            trans.wr_ack      = intf.wr_ack;
            trans.overflow    = intf.overflow;
            trans.underflow   = intf.underflow;
            trans.full        = intf.full;
            trans.empty       = intf.empty;
            trans.almostfull  = intf.almostfull;
            trans.almostempty = intf.almostempty;

            fork
            begin
                cov.sample_data(trans);
            end
            begin
                score.check_data(trans);
            end
            join

            if (test_finished) begin
            $display("Simulation finished..... Correct_counts =
, Errors_counts=%d", correct_count, error_count);
            $stop;
            end
        end
    end
endmodule

```

SCOREBOARD CLASS:

```
package scoreboard_pkg;
    import transaction_pkg::*;
    import shared_pkg::*;

    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;

    int size = 0;

    class  FIFO_scoreboard;
        logic [FIFO_WIDTH-1 : 0] data_out_ref ;

        bit [15:0] ref_mem[$];  //queue for FIFO storage

        function void check_data (input FIFO_transaction txn);
            reference_model(txn);
            if (txn.data_out !== data_out_ref) begin
                $error(" Mismatch! DUT vs REF\n"," data_out: DUT=%0h REF=%0h\n",txn.data_out, data_out_ref);
                error_count++;
            end
            else begin
                correct_count++;
            end
        endfunction

        function void reference_model (input FIFO_transaction txn);
            size = ref_mem.size();
            // Handle reset
            if (!txn.rst_n) begin
                ref_mem.delete();
                data_out_ref = 0;
                size = 0 ;
            end
            // Handle write only
            else if (txn.wr_en && !txn.rd_en) begin
                if (!(size==8)) begin
                    ref_mem.push_back(txn.data_in);
                end
            end
            // Handle read only
            else if (!txn.wr_en && txn.rd_en) begin
                if (!(size==0)) begin
                    data_out_ref = ref_mem.pop_front();
                end
            end
            // Handle READ && WRITE together
            else if (txn.wr_en && txn.rd_en) begin
                if (size==0) begin              //write only
                    ref_mem.push_back(txn.data_in);
                end
                else if (size==8) begin //read only
                    data_out_ref = ref_mem.pop_front();
                end
                else begin
                    data_out_ref = ref_mem.pop_front();
                    ref_mem.push_back(txn.data_in);
                end
            end
        endfunction
    endclass
endpackage
```

COVERAGE CLASS:

```
package coverage_pkg;
    import transaction_pkg::*;
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;

    class FIFO_coverage;
        FIFO_transaction F_cvg_txn = new();

        covergroup FIFO_cg;
            option.per_instance = 1;

            // simple coverpoints for wr_en / rd_en
            cp_wr : coverpoint F_cvg_txn.wr_en {
                bins write_0 = {0};
                bins write_1 = {1};
            }
            cp_rd : coverpoint F_cvg_txn.rd_en {
                bins read_0 = {0};
                bins read_1 = {1};
            }

            // status / event coverpoints
            cp_full : coverpoint F_cvg_txn.full {
                bins full_0 = {0};
                bins full_1 = {1};
            }

            cp_almostfull : coverpoint F_cvg_txn.almostfull {
                bins almostfull_0 = {0};
                bins almostfull_1 = {1};
            }

            cp_empty : coverpoint F_cvg_txn.empty {
                bins empty_0 = {0};
                bins empty_1 = {1};
            }

            cp_almostempty : coverpoint F_cvg_txn.almostempty {
                bins almostempty_0 = {0};
                bins almostempty_1 = {1};
            }

            // event-like coverpoints (these we will protect with ignore_bins)
            cp_overflow : coverpoint F_cvg_txn.overflow {
                bins overflow_0 = {0};
                bins overflow_1 = {1};
            }

```

```
            cp_underflow : coverpoint F_cvg_txn.underflow {
                bins underflow_0 = {0};
                bins underflow_1 = {1};
            }

            cp_wr_ack : coverpoint F_cvg_txn.wr_ack {
                bins wr_ack_0 = {0};
                bins wr_ack_1 = {1};
            }

            // ---------- crosses --------
            cross_almostfull : cross cp_wr, cp_rd, cp_almostfull;
            cross_almostempty : cross cp_wr, cp_rd, cp_almostempty;
            cross_empty : cross cp_wr, cp_rd, cp_empty;
            // full: ignore cases where full==1 but wr_en=={0,1} ,rd_en==1
            cross_full : cross cp_wr, cp_rd, cp_full {
                //full flag can not be high after reading data
                ignore_bins case1_full = binsof(cp_full.full_1) && binsof(cp_rd.read_1)&& binsof(cp_wr.write_0);
                ignore_bins case2_full = binsof(cp_full.full_1) && binsof(cp_rd.read_1)&& binsof(cp_wr.write_1);
            }
            // overflow: ignore cases where overflow==1 but wr_en==0 ,rd_en=={0,1}
            cross_overflow : cross cp_wr, cp_rd, cp_overflow {
                //overflow flag can not be high without writing new data in memory
                ignore_bins case1_overflow = binsof(cp_overflow.overflow_1) && binsof(cp_rd.read_0)&& binsof(cp_wr.write_0);
                ignore_bins case2_overflow = binsof(cp_overflow.overflow_1) && binsof(cp_rd.read_1)&& binsof(cp_wr.write_0);
            }
            // underflow: ignore cases where underflow==1 but wr_en=={0,1} ,rd_en==0
            cross_underflow : cross cp_wr, cp_rd, cp_underflow {
                //underflow flow flag can not be high without reading data from memory
                ignore_bins case1_underflow = binsof(cp_underflow.underflow_1) && binsof(cp_rd.read_0)&& binsof(cp_wr.write_0);
                ignore_bins case2_underflow = binsof(cp_underflow.underflow_1) && binsof(cp_rd.read_0)&& binsof(cp_wr.write_1);
            }
            // wr_ack: ignore cases where wr_ack==1 but wr_en==0 ,rd_en=={0,1}
            cross_wr_ack : cross cp_wr, cp_rd, cp_wr_ack {
                //wr_ack flag can not be high without writing new data in memory
                ignore_bins case1_wr_ack = binsof(cp_wr_ack.wr_ack_1) && binsof(cp_rd.read_0)&& binsof(cp_wr.write_0);
                ignore_bins case2_wr_ack = binsof(cp_wr_ack.wr_ack_1) && binsof(cp_rd.read_1)&& binsof(cp_wr.write_0);
            }
        endgroup

        // constructor
        function new();
            FIFO_cg = new();
        endfunction

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

```

TRANSACTION CLASS:

```
package transaction_pkg;
    import shared_pkg::*;
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;

    class  FIFO_transaction  ;
        logic 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 wr_ack,overflow ;
        logic full, empty, almostfull, almostempty, underflow ;

        integer RD_EN_ON_DIST ;
        integer WR_EN_ON_DIST ;

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

        //Constraint the reset to be deactivated most of the time
        constraint Reset {rst_n dist {1 := 90 , 0 := 10};} ;

        //Constraint the write enable to be high with distribution of the value
        //WR_EN_ON_DIST and to be low with 100-WR_EN_ON_DIST
        constraint write {wr_en dist {1 := WR_EN_ON_DIST , 0 := (100-WR_EN_ON_DIST)};} ;

        //Constraint the read enable the same as write enable but using RD_EN_ON_DIST
        constraint read  {rd_en dist {1 := RD_EN_ON_DIST , 0 := (100-RD_EN_ON_DIST)};} ;

    endclass
endpackage
```

TEST BENCH:

```
import transaction_pkg::*;
import shared_pkg::*;
module FIFO_tb (FIFO_if.TB intf);

    FIFO_transaction a = new();

    initial begin
        // start with reset high, set safe defaults
        //FIFO_0
        intf.rst_n     = 1;
        intf.wr_en     = 0;
        intf.rd_en     = 0;
        intf.data_in   = 0;

        // settle for a couple clocks
        repeat (2) @(negedge intf.clk);

        // apply a clean async reset pulse
        intf.rst_n = 0;
        @(negedge intf.clk);
        intf.rst_n = 1;
        @(negedge intf.clk);

        //FIFO_1
        repeat(1000) begin
            assert (a.randomize()) else $fatal("randomize failed");
            @(negedge intf.clk);
            // Drive interface from randomized transaction
            intf.rst_n     = a.rst_n;
            intf.wr_en     = a.wr_en;
            intf.rd_en     = a.rd_en;
            intf.data_in   = a.data_in;

            -> sample_event;
        end

        // Assert test_finished to stop monitor
        test_finished = 1;
    end
endmodule
```

SRC_FILES LIST:

```
shared_pkg.sv
transaction_pkg.sv
scoreboard_pkg.sv
coverage_pkg.sv
FIFO_interface.sv
FIFO.sv
monitor.sv
FIFO_tb.sv
top.sv
```

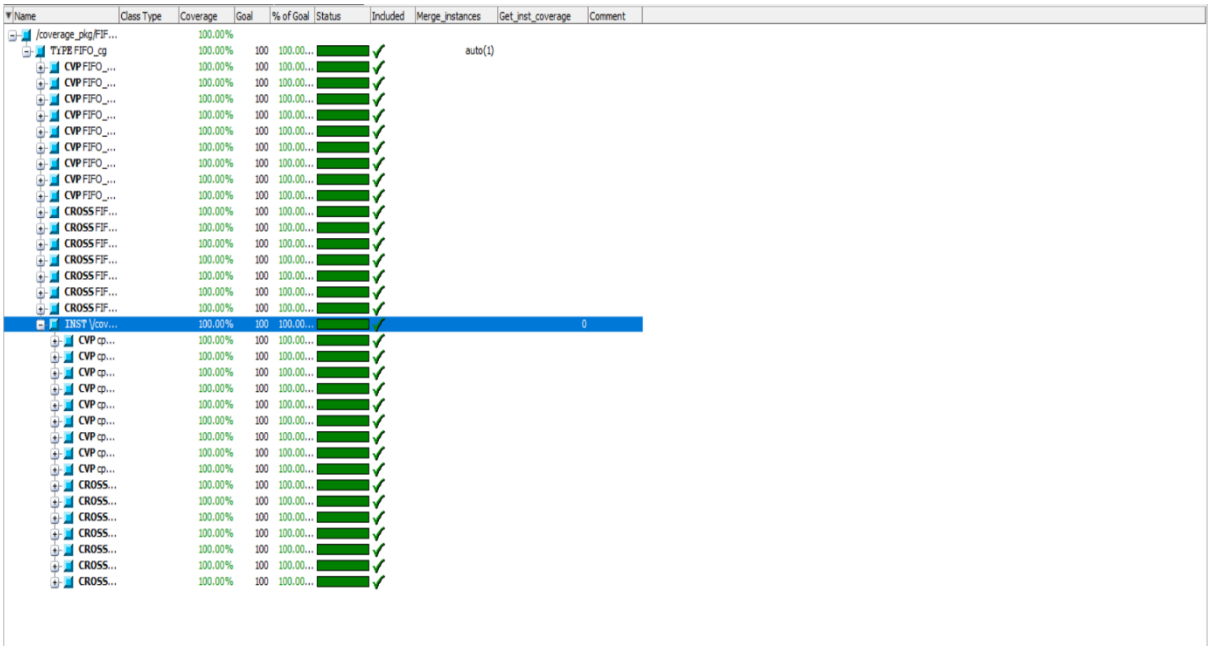
DO FILE:

```
vlib work
vlog -f src_files +define+SIM +cover -covercells
vsim -voptargs=+acc top -cover
add wave -position insertpoint sim:/top/intf/*
coverage save top.ucdb -onexit
run 0
run -all
```

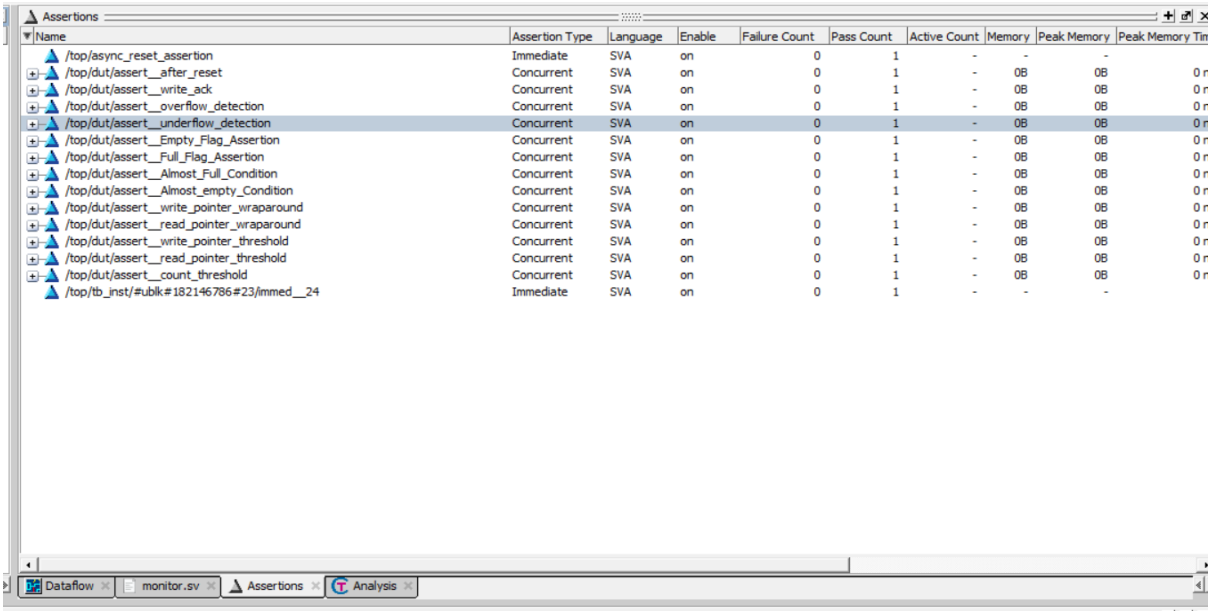
VERIFICATION PLAN:

	A	B	C	D	E
1	**Label**	**Design Requirement Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
2	FIFO_0	Reset Behavior: When the active-low reset (rst_n) is asserted, the write pointer (wr_ptr), read pointer (rd_ptr), and item count (count) must be reset to 0. All output signals reset to 0.	Directed at the start of the sim, then randomized with constraint that drive the reset to be off most of the	-	A checker in the score board to make sure the output is correct and cocurrent assertions to check internal signals
3	FIFO_1	when wr_en is high, we will write in memory data_in incase of memory is not full ,when rd_en is high data_out will take the values from memory incase of not empty memory	Randomized during simulation due to constrains to make wr_en is high 70% of time and rd_en to be high 30% of	cover all cases of wr_en, rd_en ,full and empty ,almostfull ,almostempty , wr_ack ,overflow and underflow and ignore some cases as wr_en and rd_en both low	Output Checked against refrence model in the score bord and SVA inside design file to check internal signals

FUNCTIONAL COVERAGE REPORT:



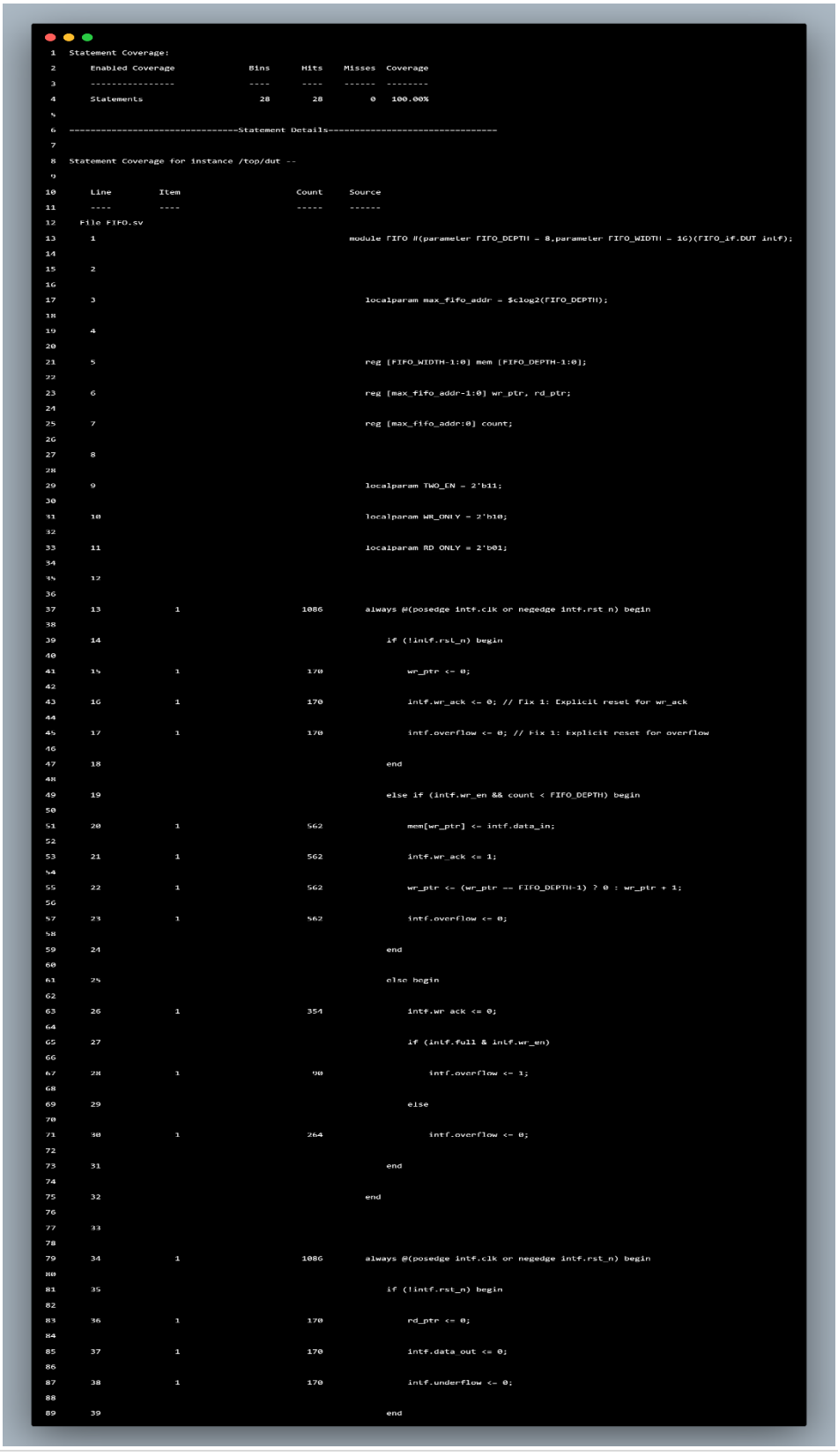
SVA COVERAGE REPORT:



```
Assertion Coverage:
    Assertions                      13        13         0   100.00%
--------------------------------------------------------------------
Name                 File(Line)                   Failure      Pass
                                                  Count        Count
--------------------------------------------------------------------
/top/dut/assert__count_threshold
                     FIFO.sv(173)                       0          1
/top/dut/assert__read_pointer_threshold
                     FIFO.sv(166)                       0          1
/top/dut/assert__write_pointer_threshold
                     FIFO.sv(159)                       0          1
/top/dut/assert__read_pointer_wraparound
                     FIFO.sv(152)                       0          1
/top/dut/assert__write_pointer_wraparound
                     FIFO.sv(144)                       0          1
/top/dut/assert__Almost_empty_Condition
                     FIFO.sv(136)                       0          1
/top/dut/assert__Almost_Full_Condition
                     FIFO.sv(128)                       0          1
/top/dut/assert__Full_Flag_Assertion
                     FIFO.sv(120)                       0          1
/top/dut/assert__Empty_Flag_Assertion
                     FIFO.sv(112)                       0          1
/top/dut/assert__underflow_detection
                     FIFO.sv(104)                       0          1
/top/dut/assert__overflow_detection
                     FIFO.sv(96)                        0          1
/top/dut/assert__write_ack
                     FIFO.sv(88)                        0          1
/top/dut/assert__after_reset
                     FIFO.sv(80)                        0          1
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        26        26         0   100.00%
```

CODE COVERAGE REPORT:

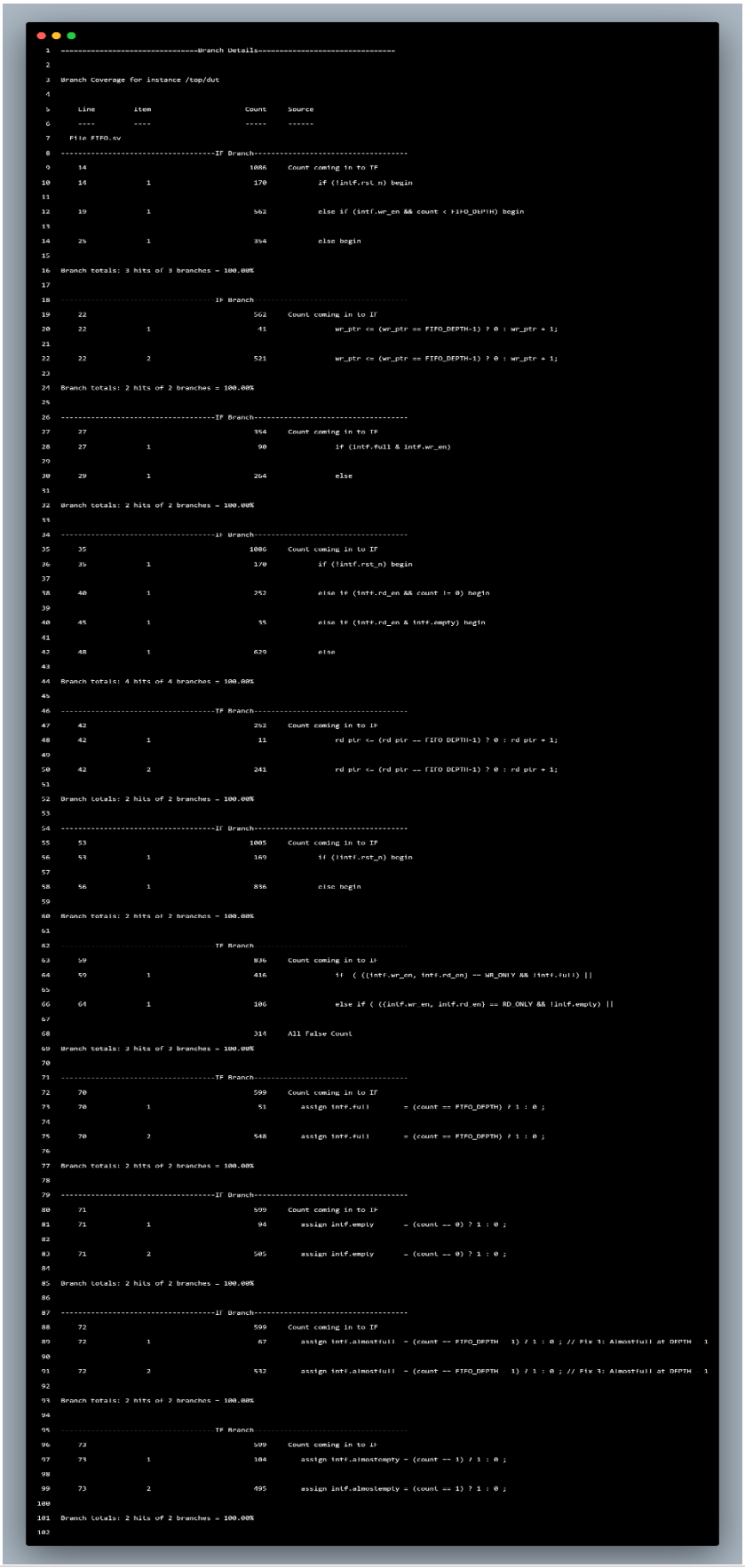
➢ Statement coverage:



```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      28        28         0   100.00%

--------------------------------Statement Details--------------------------------

Statement Coverage for instance /top/dut --

    Line         Item                    Count     Source
    ----         ----                    -----     ------
  File FIFO.sv
    1                                              module FIFO #(parameter FIFO_DEPTH = 8,parameter FIFO_WIDTH = 16)(FIFO_if.DUT intf);

    2

    3                                                 localparam max_fifo_addr = $clog2(FIFO_DEPTH);

    4

    5                                                 reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];

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

    7                                                 reg [max_fifo_addr:0] count;

    8

    9                                                 localparam TWO_EN = 2'b11;

    10                                                localparam WR_ONLY = 2'b10;

    11                                                localparam RD_ONLY = 2'b01;

    12

    13            1                    1086       always @(posedge intf.clk or negedge intf.rst_n) begin

    14                                                    if (!intf.rst_n) begin

    15            1                     170               wr_ptr <= 0;

    16            1                     170               intf.wr_ack <= 0; // Fix 1: Explicit reset for wr_ack

    17            1                     170               intf.overflow <= 0; // Fix 1: Explicit reset for overflow

    18                                                    end

    19                                                    else if (intf.wr_en && count < FIFO_DEPTH) begin

    20            1                     562               mem[wr_ptr] <= intf.data_in;

    21            1                     562               intf.wr_ack <= 1;

    22            1                     562               wr_ptr <= (wr_ptr == FIFO_DEPTH-1) ? 0 : wr_ptr + 1;

    23            1                     562               intf.overflow <= 0;

    24                                                    end

    25                                                    else begin

    26            1                     354               intf.wr_ack <= 0;

    27                                                    if (intf.full & intf.wr_en)

    28            1                      90                   intf.overflow <= 1;

    29                                                    else

    30            1                     264                   intf.overflow <= 0;

    31                                                end

    32                                            end

    33

    34            1                    1086       always @(posedge intf.clk or negedge intf.rst_n) begin

    35                                                    if (!intf.rst_n) begin

    36            1                     170               rd_ptr <= 0;

    37            1                     170               intf.data_out <= 0;

    38            1                     170               intf.underflow <= 0;

    39                                                end
```

```
46            1                    35              intf.underflow <= 1;

47                                                 end

48                                                 else

49            1                    629                 intf.underflow <= 0;

50                                               end

51

52            1                    1005        always @(posedge intf.clk or negedge intf.rst_n) begin

53                                                 if (!intf.rst_n) begin

54            1                    169                 count <= 0;

55                                                 end

56                                                 else begin

57                                                     // Fix 2: Correctly handle simultaneous R/W on boundary conditions.

58                                                     // Count + 1: Write-only or R/W on empty (Write takes precedence)

59                                                     if  ( ({intf.wr_en, intf.rd_en} == WR_ONLY && !intf.full) ||

60                                                           ({intf.wr_en, intf.rd_en} == TWO_EN && intf.empty) )

61            1                    416                    count <= count + 1;

62

63                                                      // Count - 1: Read-only or R/W on full (Read takes precedence)

64                                                     else if ( ({intf.wr_en, intf.rd_en} == RD_ONLY && !intf.empty) ||

65                                                               ({intf.wr_en, intf.rd_en} == TWO_EN && intf.full) )

66            1                    106                    count <= count - 1;

67                                                 end

68                                               end

69

70            1                    600         assign intf.full        = (count == FIFO_DEPTH) ? 1 : 0 ;

71            1                    600         assign intf.empty       = (count == 0) ? 1 : 0 ;

72            1                    600         assign intf.almostfull  = (count == FIFO_DEPTH - 1) ? 1 : 0 ; // Fix 3: Almostfull at DEPTH - 1

73            1                    600         assign intf.almostempty = (count == 1) ? 1 : 0 ;

```

➢ Branch coverage:



```
1   ------------------------------Branch Details------------------------------
2
3   Branch Coverage for instance /top/dut
4
5       Line         Item                     Count    Source
6       ----         ----                     -----    ------
7     File FIFO.sv
8   ------------------------------------IF Branch------------------------------------
9       14                                   1086     Count coming in to IF
10      14             1                     170              if (!intf.rst_n) begin
11
12      19             1                     562              else if (intf.wr_en && count < FIFO_DEPTH) begin
13
14      25             1                     354              else begin
15
16  Branch totals: 3 hits of 3 branches = 100.00%
17
18                                      IF Branch                                      
19      22                                   562     Count coming in to IF
20      22             1                      41                  wr_ptr <= (wr_ptr == FIFO_DEPTH-1) ? 0 : wr_ptr + 1;
21
22      22             2                     521                  wr_ptr <= (wr_ptr == FIFO_DEPTH-1) ? 0 : wr_ptr + 1;
23
24  Branch totals: 2 hits of 2 branches = 100.00%
25
26  ------------------------------------IF Branch------------------------------------
27      27                                   354     Count coming in to IF
28      27             1                      90                  if (intf.full & intf.wr_en)
29
30      29             1                     264                  else
31
32  Branch totals: 2 hits of 2 branches = 100.00%
33
34  ------------------------------------If Branch------------------------------------
35      35                                  1086     Count coming in to IF
36      35             1                     170              if (!intf.rst_n) begin
37
38      40             1                     252              else if (intf.rd_en && count != 0) begin
39
40      45             1                      95              else if (intf.rd_en & intf.empty) begin
41
42      48             1                     629              else
43
44  Branch totals: 4 hits of 4 branches = 100.00%
45
46  ------------------------------------TF Branch------------------------------------
47      42                                   252     Count coming in to IF
48      42             1                      11                  rd_ptr <= (rd_ptr == FIFO_DEPTH-1) ? 0 : rd_ptr + 1;
49
50      42             2                     241                  rd_ptr <= (rd_ptr == FIFO_DEPTH-1) ? 0 : rd_ptr + 1;
51
52  Branch totals: 2 hits of 2 branches = 100.00%
53
54  ------------------------------------IF Branch------------------------------------
55      53                                  1005     Count coming in to IF
56      53             1                     169              if (!intf.rst_n) begin
57
58      56             1                     836              else begin
59
60  Branch totals: 2 hits of 2 branches = 100.00%
61
62                                      TF Branch                                      
63      59                                   836     Count coming in to IF
64      59             1                     416                  if  ( ((intf.wr_en, intf.rd_en) == WR_ONLY && !intf.full) ||
65
66      64             1                     106                  else if ( ((intf.wr_en, intf.rd_en) == RD_ONLY && !intf.empty) ||
67
68                                           314     All false Count
69  Branch totals: 3 hits of 3 branches = 100.00%
70
71  ------------------------------------TF Branch------------------------------------
72      70                                   599     Count coming in to IF
73      70             1                      51        assign intf.full        = (count == FIFO_DEPTH) ? 1 : 0 ;
74
75      70             2                     548        assign intf.full        = (count == FIFO_DEPTH) ? 1 : 0 ;
76
77  Branch totals: 2 hits of 2 branches = 100.00%
78
79  ------------------------------------IF Branch------------------------------------
80      71                                   599     Count coming in to IF
81      71             1                      94        assign intf.empty       = (count == 0) ? 1 : 0 ;
82
83      71             2                     505        assign intf.empty       = (count == 0) ? 1 : 0 ;
84
85  Branch totals: 2 hits of 2 branches = 100.00%
86
87  ------------------------------------IF Branch------------------------------------
88      72                                   599     Count coming in to IF
89      72             1                      67        assign intf.almostfull  = (count == FIFO_DEPTH   1) ? 1 : 0 ; // Fix 3: Almostfull at DEPTH   1
90
91      72             2                     532        assign intf.almostfull  = (count == FIFO_DEPTH   1) ? 1 : 0 ; // Fix 3: Almostfull at DEPTH   1
92
93  Branch totals: 2 hits of 2 branches = 100.00%
94
95                                      TF Branch                                      
96      73                                   599     Count coming in to IF
97      73             1                     104        assign intf.almostempty = (count == 1) ? 1 : 0 ;
98
99      73             2                     495        assign intf.almostempty = (count == 1) ? 1 : 0 ;
100
101 Branch totals: 2 hits of 2 branches = 100.00%
102
```

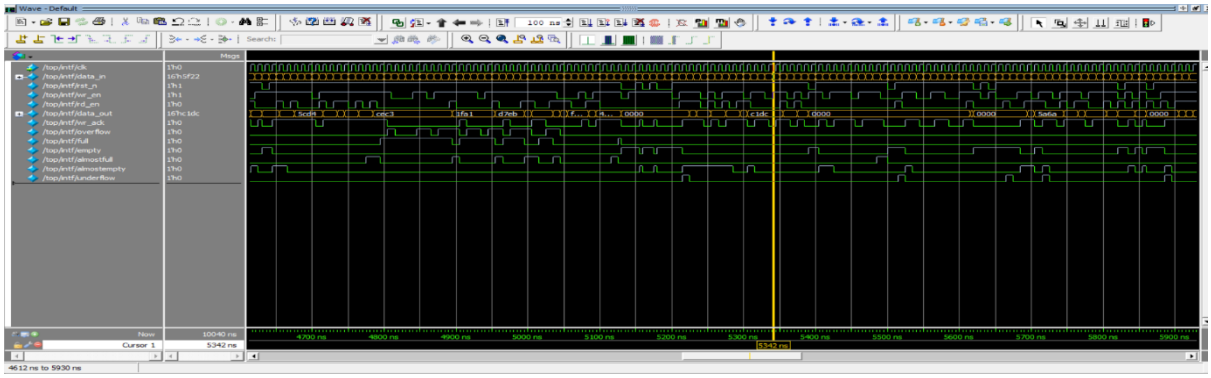
➢ Toggle coverage:

```
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         86        86         0   100.00%

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

Toggle Coverage for instance /top/intf --

                                          Node      1H->0L      0L->1H  "Coverage"
                                          ----------------------------------------
                                   almostempty           1           1      100.00
                                    almostfull           1           1      100.00
                                           clk           1           1      100.00
                                  data_in[15-0]          1           1      100.00
                                 data_out[15-0]          1           1      100.00
                                         empty           1           1      100.00
                                          full           1           1      100.00
                                      overflow           1           1      100.00
                                         rd_en           1           1      100.00
                                         rst_n           1           1      100.00
                                     underflow           1           1      100.00
                                        wr_ack           1           1      100.00
                                         wr_en           1           1      100.00

Total Node Count     =         43
Toggled Node Count   =         43
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (86 of 86 bins)

================================================================================
```

Questa sim snippets:

```
#
# Top level modules:
#       top
# End time: 18:05:31 on Oct 02,2025, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# vsim -voptargs="+acc" top -coverage
# Start time: 18:05:31 on Oct 02,2025
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
# Loading sv_std.std
# Loading work.top(fast)
# Loading work.FIFO_if(fast__1)
# Loading work.FIFO(fast)
# Loading work.shared_pkg(fast)
# Loading work.transaction_pkg(fast)
# Loading work.FIFO_tb_sv_unit(fast)
# Loading work.FIFO_tb(fast)
# Loading work.scoreboard_pkg(fast)
# Loading work.coverage_pkg(fast)
# Loading work.monitor_sv_unit(fast)
# Loading work.monitor(fast)
# Simulation finished..... Correct_counts = 1000, Errors_counts =          0
# ** Note: $stop    : monitor.sv(42)
#    Time: 10040 ns  Iteration: 1  Instance: /top/monitor_inst
# Break in Module monitor at monitor.sv line 42
```

- **Reset:**



- **Waveforms:**