Faculty of Engineering
Cairo University

Project

Submitted to: Eng. Kareem Waseem

Eng. Abdelrahman Sherif

Submitted by: Seif Eldin Ahmed Hassan

Verification Plan:

Label	Description	Stimulus Generation	Functional Coverage (Later)	Functionality Check
FIFO_1	When the reset is asserted, the internal pointers, counters, overflow and underflow	Directed at the start of the simulation, then randomized with constraint that drive the reset to be off most of the simulation time	-	A checker in the scoreboard to check the internal signals and counter - Other signals are checked with assertions in the DUT module
FIFO_2	Verify the functionality of the FIFO with constrained randomization	Randomization in a repeat 9999 iterations with constraints on the reset to be off 90% of the time, write enable signal to be high 70% of the time, read enable signal to be active 30% of the time.	Cross coverage between signals write enable, read enable and each output control signals	A checker in the scoreboard to check the internal signals and counter - Other signals are checked with assertions in the DUT module

Design Code:

```
module FIFO(FIFO_it.DUT FIFOit);
localparam max_fifo_addr = $clog2(FIFOit.FIFO_DEPTH);
reg [FIFOit.FIFO_WIDTH-1:0] mem [FIFOit.FIFO_DEPTH-1:0];
reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [max_fifo_addr:0] count;
always @(posedge FIFOit.clk or negedge FIFOit.rst_n) begin
    if (!FIFOit.rst_n) begin
        wr_ptr <= 0;
    end
    else if (FIFOit.wr_en && count < FIFOit.FIFO_DEPTH) begin
        mem[wr_ptr] <= FIFOit.data_in;
        // FIFOit.wr_ack <= 1; // BUG HERE
        wr_ptr <= wr_ptr + 1;
    end
    // BUG HERE
    // else begin
    //  FIFOit.wr_ack <= 0;
    //  if (FIFOit.full & FIFOit.wr_en)
    //      FIFOit.overflow <= 1;
    //  else
    //      FIFOit.overflow <= 0;
    // end
end
always @(posedge FIFOit.clk or negedge FIFOit.rst_n) begin
    if (!FIFOit.rst_n) begin
        rd_ptr <= 0;
    end
    else if (FIFOit.rd_en && count != 0) begin
        FIFOit.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
end
always @(posedge FIFOit.clk or negedge FIFOit.rst_n) begin
    if (!FIFOit.rst_n) begin
        count <= 0;
    end
    else begin
        if  ((({FIFOit.wr_en, FIFOit.rd_en} == 2'b10) && !FIFOit.full) ||
        (({FIFOit.wr_en, FIFOit.rd_en} == 2'b11) && FIFOit.empty)) // Added the part for the note
            count <= count + 1;
        else if ((({FIFOit.wr_en, FIFOit.rd_en} == 2'b01) && !FIFOit.empty) ||
        (({FIFOit.wr_en, FIFOit.rd_en} == 2'b11) && FIFOit.full)) // Added the part for the note
            count <= count - 1;
    end
end
assign FIFOit.full = (count == FIFOit.FIFO_DEPTH)? 1 : 0;
assign FIFOit.empty = (count == 0)? 1 : 0;
assign FIFOit.underflow = (FIFOit.empty && FIFOit.rd_en)? 1 : 0;
// assign FIFOit.almostfull = (count == FIFOit.FIFO_DEPTH-2)? 1 : 0; // BUG HERE
assign FIFOit.almostempty = (count == 1)? 1 : 0;
// Fixed bugs
assign FIFOit.wr_ack = ((count != FIFOit.FIFO_DEPTH) && FIFOit.wr_en)? 1 : 0;
assign FIFOit.almostfull = (count == FIFOit.FIFO_DEPTH - 1)? 1 : 0;
assign FIFOit.overflow = (FIFOit.full && FIFOit.wr_en)? 1 : 0;
```

- Comment: All bugs are commented in the design snippet

Assertions:

```
8   module FIFO(FIFO_it.DUT FIFOit);
63  `ifdef SIM
64      property write_count;
65          @(posedge FIFOit.clk) disable iff(!FIFOit.rst_n)
66          (!FIFOit.rd_en && FIFOit.wr_en && count != FIFOit.FIFO_DEPTH) |=> ($past(count) + 1'b1 == count);
67      endproperty
68      property read_
                FIFO_it FIFOit (clk);
69          @(posedge FIFOit.clk) disable iff(!FIFOit.rst_n)
70          (FIFOit.rd_en && !FIFOit.wr_en && count != 0) |=> ($past(count) - 1'b1 == count);
71      endproperty
72      property read_write_count;
73          @(posedge FIFOit.clk) disable iff(!FIFOit.rst_n)
74          (FIFOit.rd_en && FIFOit.wr_en && count != 0 && count != FIFOit.FIFO_DEPTH) |=> ($past(count) == count);
75      endproperty
76      property read_write_count_empty;
77          @(posedge FIFOit.clk) disable iff(!FIFOit.rst_n)
78          (FIFOit.rd_en && FIFOit.wr_en && FIFOit.empty) |=> ($past(count) + 1'b1 == count);
79      endproperty
80      property read_write_count_full;
81          @(posedge FIFOit.clk) disable iff(!FIFOit.rst_n)
82          (FIFOit.rd_en && FIFOit.wr_en && FIFOit.full) |=> ($past(count) - 1'b1 == count);
83      endproperty
84      property write_ptr;
85          @(posedge FIFOit.clk) disable iff (!FIFOit.rst_n)
86          (FIFOit.wr_en && count != FIFOit.FIFO_DEPTH) |=> ($past(wr_ptr) + 1'b1 == wr_ptr);
87      endproperty
88      property read_ptr;
89          @(posedge FIFOit.clk) disable iff (!FIFOit.rst_n)
90          (FIFOit.rd_en && count != 0) |=> ($past(rd_ptr) + 1'b1 == rd_ptr);
91      endproperty
92
93      // Assertions
94      wr_count_assert: assert property (write_count)
95          else $error("Assertion failed: count did not increment when write correctly at time %0t, count = %0d, ex
96                      $time, count, $past(count) + 1'b1);
97      rd_count_assert: assert property (read_count)
98          else $error("Assertion failed: count did not decrement when read correctly at time %0t, count = %0d, exp
99                      $time, count, $past(count) - 1'b1);
100     rd_wr_count_assert: assert property (read_write_count)
101         else $error("Assertion failed: count should remain the same when read and write, and not empty or full a
102                     $time, count, $past(count));
103     rd_wr_count_empty_assert:assert property (read_write_count_empty)
104         else $error("Assertion failed: count did not increment when write, read and empty correctly at time %0t,
105                     $time, count, $past(count) + 1'b1);
106     rd_wr_count_full_assert:assert property (read_write_count_full)
107         else $error("Assertion failed: count did not increment when write, read and full correctly at time %0t,
108                     $time, count, $past(count) - 1'b1);
109     wr_ptr_assert: assert property (write_ptr)
110         else $error("Assertion failed: wr_ptr did not increment correctly at time %0t, wr_ptr = %0d, expected =
111                     $time, wr_ptr, $past(wr_ptr) + 1'b1);
112     rd_ptr_assert: assert property (read_ptr)
113         else $error("Assertion failed: rd_ptr did not increment correctly at time %0t, rd_ptr = %0d, expected =
114                     $time, rd_ptr, $past(rd_ptr) + 1'b1);
115
```

```
8   module FIFO(FIFO_it.DUT FIFOit);
112     rd_ptr_assert: assert property (read_ptr)
113         else $error("Assertion failed: rd_ptr did not increment correctly at time %0t, rd_ptr = %0d, expected =
114                     $time, rd_ptr, $past(rd_ptr) + 1'b1);
115
116     // Cover Assertions
117     wr_count_cover: cover property (write_count);
118     rd_count_cover: cover property (read_count);
119     rd_wr_count_cover: cover property (read_write_count);
120     rd_wr_count_empty_cover:cover property (read_write_count_empty);
121     rd_wr_count_full_cover:cover property (read_write_count_full);
122     wr_ptr_cover: cover property (write_ptr);
123     rd_ptr_cover: cover property (read_ptr);
124
125     always_comb begin
126         if(!FIFOit.rst_n)begin
127             rst_count_assert: assert final (count==0);
128             rst_count_cover: cover final (count==0);
129
130             rst_wr_ptr_assert: assert final (wr_ptr==0);
131             rst_wr_ptr_cover: cover final (wr_ptr==0);
132
133             rst_rd_ptr_assert: assert final (rd_ptr==0);
134             rst_rd_ptr_cover: cover final (rd_ptr==0);
135         end
136         if(count == FIFOit.FIFO_DEPTH) begin
137             full_assert: assert final (FIFOit.full);
138             full_cover: cover final (FIFOit.full);
139         end
140         if(count == 0) begin
141             empty_assert: assert final (FIFOit.empty);
142             empty_cover: cover final (FIFOit.empty);
143         end
144         if(count == 0 && FIFOit.rd_en) begin
145             underflow_assert: assert final (FIFOit.underflow);
146             underflow_cover: cover final (FIFOit.underflow);
147         end
148         if(count == FIFOit.FIFO_DEPTH && FIFOit.wr_en) begin
149             overflow_assert: assert final (FIFOit.overflow);
150             overflow_cover: cover final (FIFOit.overflow);
151         end
152         if(count == FIFOit.FIFO_DEPTH - 1) begin
153             almostfull_assert: assert final (FIFOit.almostfull);
154             almostfull_cover: cover final (FIFOit.almostfull);
155         end
156         if(count == 1) begin
157             almostempty_assert: assert final (FIFOit.almostempty);
158             almostempty_cover: cover final (FIFOit.almostempty);
159         end
160         if((count != FIFOit.FIFO_DEPTH) && FIFOit.wr_en) begin
161             wr_ack_assert: assert final (FIFOit.wr_ack);
162             wr_ack_cover: cover final (FIFOit.wr_ack);
163         end
164     end
165 `endif
```

Top Code:

```
1   module top;
2     bit clk;
3     always #5 clk = ~clk;
4     FIFO_it FIFOit (clk);
5     FIFO dut (FIFOit);
6     FIFO_test TEST (FIFOit);
7     FIFO_monitor monitor (FIFOit);
8   endmodule
```

Testbench Code:

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

module FIFO_test (
    FIFO_it.TEST FIFOit
);
  FIFO_transaction trans;

  initial begin
    trans = new();

    // FIFO_1
    assert_reset;

    // FIFO_2
    repeat (9999) begin
      assert (trans.randomize());
      FIFOit.data_in = trans.data_in;
      FIFOit.wr_en   = trans.wr_en;
      FIFOit.rd_en   = trans.rd_en;
      FIFOit.rst_n   = trans.rst_n;
      @(negedge FIFOit.clk);
    end
    shared_pkg::test_finished = 1;
  end

  task assert_reset;
    FIFOit.rst_n = 0;
    @(negedge FIFOit.clk);
    FIFOit.rst_n = 1;
  endtask
endmodule
```

Monitor Code:

```
import FIFO_transaction_pkg::*;
import FIFO_scoreboard_pkg::*;
import FIFO_coverage_pkg::*;
import shared_pkg::*;
module FIFO_monitor (FIFO_it.monitor FIFOit);
    FIFO_transaction f_txn;
    FIFO_scoreboard f_scoreboard = new();
    FIFO_coverage f_coverage = new();
    initial begin
        f_txn = new();
        forever begin
            f_txn.data_in = FIFOit.data_in;
            f_txn.wr_en = FIFOit.wr_en;
            f_txn.rd_en = FIFOit.rd_en;
            f_txn.rst_n = FIFOit.rst_n;
            @(negedge FIFOit.clk);
            f_txn.data_out = FIFOit.data_out;
            f_txn.full = FIFOit.full;
            f_txn.almostfull = FIFOit.almostfull;
            f_txn.empty = FIFOit.empty;
            f_txn.almostempty = FIFOit.almostempty;
            f_txn.overflow = FIFOit.overflow;
            f_txn.underflow = FIFOit.underflow;
            f_txn.wr_ack = FIFOit.wr_ack;
            @(posedge FIFOit.clk);

            fork
                begin
                    f_coverage.sample_data(f_txn);
                end

                begin
                    f_scoreboard.check_data(f_txn);
                end
            join

            if (shared_pkg::test_finished) begin
                $display("--------------------Summary--------------------");
                $display("Test finished. Correct: %0d, Errors: %0d", correct_count, error_count);
                $stop;
            end
        end
    end
endmodule
```

Interface Code:

```
interface FIFO_it (
    input bit clk
);
  parameter FIFO_WIDTH = 16;
  parameter FIFO_DEPTH = 8;

  // Input Signals
  logic [FIFO_WIDTH-1:0] data_in;
  logic rst_n, wr_en, rd_en;

  // Output signals
  logic [FIFO_WIDTH-1:0] data_out;
  logic wr_ack, overflow;
  logic full, empty, almostfull, almostempty, underflow;

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

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

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

Coverage Code:

```
package FIFO_coverage_pkg;
    import FIFO_transaction_pkg::*;
    class FIFO_coverage;
        FIFO_transaction F_cvg_txn;
        covergroup cg_FIFO;
            cp_wr_en: coverpoint F_cvg_txn.wr_en;
            cp_rd_en: coverpoint F_cvg_txn.rd_en;
            cp_wr_ack: coverpoint F_cvg_txn.wr_ack;
            cp_overflow: coverpoint F_cvg_txn.overflow;
            cp_full: coverpoint F_cvg_txn.full;
            cp_empty: coverpoint F_cvg_txn.empty;
            cp_almostfull: coverpoint F_cvg_txn.almostfull;
            cp_almostempty: coverpoint F_cvg_txn.almostempty;
            cp_underflow: coverpoint F_cvg_txn.underflow;

            // Cross coverage
            // Write Ack cross with ignored bins
            cx_wr_ack: cross cp_wr_en, cp_rd_en, cp_wr_ack {
                ignore_bins auto_wr_ack_0 = binsof(cp_wr_en) intersect {0} &&
                                            binsof(cp_rd_en) intersect {1} &&
                                            binsof(cp_wr_ack) intersect {1};
                ignore_bins auto_wr_ack_1 = binsof(cp_wr_en) intersect {0} &&
                                            binsof(cp_rd_en) intersect {0} &&
                                            binsof(cp_wr_ack) intersect {1};
                ignore_bins auto_wr_ack_2 = binsof(cp_wr_en) intersect {1} &&
                                            binsof(cp_rd_en) intersect {1} &&
                                            binsof(cp_wr_ack) intersect {0};
            }
            // Overflow cross with ignored bins
            cx_overflow: cross cp_wr_en, cp_rd_en, cp_overflow {
                ignore_bins auto_overflow_0 = binsof(cp_wr_en) intersect {0} &&
                                              binsof(cp_rd_en) intersect {1} &&
                                              binsof(cp_overflow) intersect {1};
                ignore_bins auto_overflow_1 = binsof(cp_wr_en) intersect {0} &&
                                              binsof(cp_rd_en) intersect {0} &&
                                              binsof(cp_overflow) intersect {1};
                ignore_bins auto_overflow_2 = binsof(cp_wr_en) intersect {1} &&
                                              binsof(cp_rd_en) intersect {1} &&
                                              binsof(cp_overflow) intersect {1};
            }
            // Full cross with ignored bins
            cx_full: cross cp_wr_en, cp_rd_en, cp_full {
                ignore_bins auto_full_0 = binsof(cp_wr_en) intersect {0} &&
                                          binsof(cp_rd_en) intersect {1} &&
                                          binsof(cp_full) intersect {1};
```

```
package FIFO_coverage_pkg;
    class FIFO_coverage;
            // Full cross with ignored bins
            cx_full: cross cp_wr_en, cp_rd_en, cp_full {
                ignore_bins auto_full_0 = binsof(cp_wr_en) intersect {0} &&
                                          binsof(cp_rd_en) intersect {1} &&
                                          binsof(cp_full) intersect {1};
                ignore_bins auto_full_1 = binsof(cp_wr_en) intersect {1} &&
                                          binsof(cp_rd_en) intersect {1} &&
                                          binsof(cp_full) intersect {1};
            }
            // Empty cross coverage
            cx_empty: cross cp_wr_en, cp_rd_en, cp_empty;

            // Almost full cross coverage
            cx_almostfull: cross cp_wr_en, cp_rd_en, cp_almostfull;

            // Almost empty cross coverage
            cx_almostempty: cross cp_wr_en, cp_rd_en, cp_almostempty;

            // Underflow cross with ignored bins
            cx_underflow: cross cp_wr_en, cp_rd_en, cp_underflow {
                ignore_bins auto_underflow_0 = binsof(cp_wr_en) intersect {1} &&
                                               binsof(cp_rd_en) intersect {0} &&
                                               binsof(cp_underflow) intersect {1};
                ignore_bins auto_underflow_1 = binsof(cp_wr_en) intersect {0} &&
                                               binsof(cp_rd_en) intersect {0} &&
                                               binsof(cp_underflow) intersect {1};
            }
        endgroup: cg_FIFO

        function new;
            cg_FIFO = new;
            cg_FIFO.start();
        endfunction

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

Scoreboard Code:

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

  class FIFO_scoreboard #(
      FIFO_WIDTH = 16,
      FIFO_DEPTH = 8
  );
    localparam max_fifo_addr = $clog2(FIFO_DEPTH);

    bit [FIFO_WIDTH-1:0] data_out_ref;

    bit [FIFO_WIDTH - 1 : 0] fifo_mem[FIFO_DEPTH-1:0];

    bit [max_fifo_addr - 1 : 0] write_ptr, read_ptr;
    bit [max_fifo_addr : 0] count;

    function void reference_model(FIFO_transaction trans_obj);
        if (!trans_obj.rst_n) begin
            read_ptr = 0;
            write_ptr = 0;
            count = 0;
        end
        else begin
            if (trans_obj.rd_en && count != 0) begin
                data_out_ref = fifo_mem[read_ptr];
                read_ptr++;
            end
            if (trans_obj.wr_en && count != FIFO_DEPTH) begin
                fifo_mem[write_ptr] = trans_obj.data_in;
                write_ptr++;
            end
            if((trans_obj.wr_en && !trans_obj.rd_en && count != FIFO_DEPTH) ||
            (trans_obj.wr_en && trans_obj.rd_en && count == 0))begin
                count++;
            end
            if((!trans_obj.wr_en && trans_obj.rd_en && count != 0) ||
            (trans_obj.wr_en && trans_obj.rd_en && count == FIFO_DEPTH)) begin
                count--;
            end
        end
    endfunction

    function void check_data(FIFO_transaction trans_obj);
        reference_model(trans_obj);
```

```
package FIFO_scoreboard_pkg;
  class FIFO_scoreboard #(
                write_ptr++;
            end
            if((trans_obj.wr_en && !trans_obj.rd_en && count != FIFO_DEPTH) ||
            (trans_obj.wr_en && trans_obj.rd_en && count == 0))begin
                count++;
            end
            if((!trans_obj.wr_en && trans_obj.rd_en && count != 0) ||
            (trans_obj.wr_en && trans_obj.rd_en && count == FIFO_DEPTH)) begin
                count--;
            end
        end
    endfunction

    function void check_data(FIFO_transaction trans_obj);
        reference_model(trans_obj);
        if(trans_obj.data_out != data_out_ref) begin
            $display("Error at time = %0t: data_out mismatch: Expected = %0h, Got = %0h",
                     $time, data_out_ref, trans_obj.data_out);
            shared_pkg::error_count++;
        end else begin
            $display("Correct at time = %0t!", $time);
            shared_pkg::correct_count++;
        end
    endfunction
  endclass
endpackage
```

Transaction Code:

```
package FIFO_transaction_pkg;
  parameter FIFO_WIDTH = 16;
  class FIFO_transaction;
    rand bit [FIFO_WIDTH-1:0] data_in;
    rand bit rst_n, wr_en, rd_en;
    bit [FIFO_WIDTH-1:0] data_out;
    bit wr_ack, overflow;
    bit full, empty, almostfull, almostempty, underflow;

    int RD_EN_ON_DIST;
    int WR_EN_ON_DIST;

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

    constraint reset_con {
      rst_n dist {
        0 :/ 10,
        1 :/ 90
      };
    }

    constraint wr_en_con {
      wr_en dist {
        1 :/ WR_EN_ON_DIST,
        0 :/ (100 - WR_EN_ON_DIST)
      };
    }

    constraint rd_en_con {
      rd_en dist {
        1 :/ RD_EN_ON_DIST,
        0 :/ (100 - RD_EN_ON_DIST)
      };
    }
  endclass
endpackage
```

Shared Package:

```
package shared_pkg;
    int error_count = 0;
    int correct_count = 0;
    bit test_finished;
endpackage
```

src_file:

```
FIFO.sv
shared_pkg.sv
FIFO_transaction.sv
FIFO_scoreboard.sv
FIFO_coverage.sv
FIFO_it.sv
FIFO_test.sv
FIFO_monitor.sv
top.sv
```

Do File:

```
vlib work
vlog -f src_files.list -mfcu +define+SIM +cover
vsim -voptargs=+acc work.top -cover
add wave /top/FIFOit/*
coverage save fifocoveragereport.ucdb -onexit -du work.top
run -all
```

Transcript before fixing bugs:

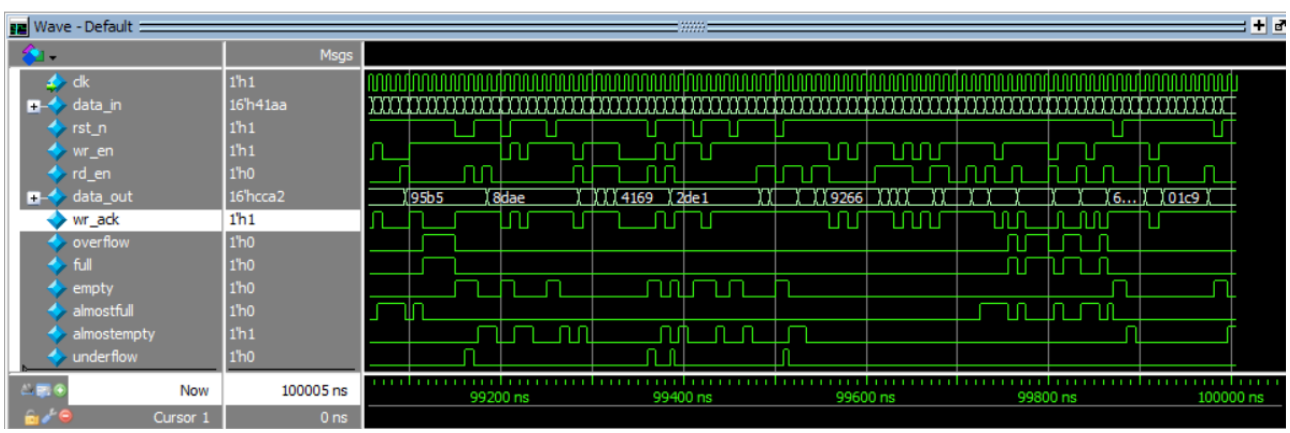
```
# Error at time = 99795: data_out mismatch: Expected = 9fc2, Got = 1075
# Error at time = 99805: data_out mismatch: Expected = 9fc2, Got = 1075
# ** Error: Assertion error.
#    Time: 99805 ns  Scope: top.dut.almostfull_assert File: FIFO.sv Line: 151
# ** Error: Assertion error.
#    Time: 99810 ns  Scope: top.dut.wr_ack_assert File: FIFO.sv Line: 159
# ** Error: Assertion error.
#    Time: 99810 ns  Scope: top.dut.almostfull_assert File: FIFO.sv Line: 151
# Error at time = 99815: data_out mismatch: Expected = 69d9, Got = 5882
# ** Error: Assertion error.
#    Time: 99815 ns  Scope: top.dut.overflow_assert File: FIFO.sv Line: 147
# Error at time = 99825: data_out mismatch: Expected = 69d9, Got = 5882
# Error at time = 99835: data_out mismatch: Expected = 69d9, Got = 5882
# Error at time = 99845: data_out mismatch: Expected = 5d8, Got = cdfd
# ** Error: Assertion failed: count did not increment when write, read and full correctly at time 99845, count = 8, expected = 7
#    Time: 99845 ns Started: 99835 ns  Scope: top.dut.rd_wr_count_full_assert File: FIFO.sv Line: 105
# ** Error: Assertion error.
#    Time: 99850 ns  Scope: top.dut.overflow_assert File: FIFO.sv Line: 147
# Error at time = 99855: data_out mismatch: Expected = 5d8, Got = cdfd
# Error at time = 99865: data_out mismatch: Expected = 5d8, Got = cdfd
# ** Error: Assertion error.
#    Time: 99870 ns  Scope: top.dut.wr_ack_assert File: FIFO.sv Line: 159
# Error at time = 99875: data_out mismatch: Expected = 6e78, Got = c908
# ** Error: Assertion error.
#    Time: 99880 ns  Scope: top.dut.wr_ack_assert File: FIFO.sv Line: 159
# Error at time = 99885: data_out mismatch: Expected = 6e78, Got = c908
# Error at time = 99895: data_out mismatch: Expected = 6e78, Got = c908
# Error at time = 99905: data_out mismatch: Expected = 6e78, Got = c908
# Correct at time = 99915!
# ** Error: Assertion error.
#    Time: 99920 ns  Scope: top.dut.wr_ack_assert File: FIFO.sv Line: 159
# Correct at time = 99925!
# Correct at time = 99935!
# Correct at time = 99945!
# Correct at time = 99955!
# Correct at time = 99965!
# Correct at time = 99975!
# Correct at time = 99985!
# Correct at time = 99995!
# Correct at time = 100005!
# --------------------Summary--------------------
# Test finished. Correct: 8132, Errors: 1868
# ** Note: $stop    : FIFO_monitor.sv(40)
#    Time: 100005 ns  Iteration: 1  Instance: /top/monitor
# Break in Module FIFO_monitor at FIFO_monitor.sv line 40

VSIM 2>
```

Transcript after fixing bugs:

```
# ---------------------Summary---------------------
# Test finished. Correct: 10000, Errors: 0
# ** Note: $stop    : FIFO_monitor.sv(40)
#    Time: 100005 ns  Iteration: 1  Instance: /top/monitor
```

Waveform:



Assertions:

Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Pea
/top/dut/wr_count_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/top/dut/rd_count_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/top/dut/rd_wr_count_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/top/dut/rd_wr_count_empty_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/top/dut/rd_wr_count_full_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/top/dut/wr_ptr_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/top/dut/rd_ptr_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/top/dut/rst_count_assert	Immediate	SVA	on	0	1	-	-	-	
/top/dut/rst_wr_ptr_assert	Immediate	SVA	on	0	1	-	-	-	
/top/dut/rst_rd_ptr_assert	Immediate	SVA	on	0	1	-	-	-	
/top/dut/full_assert	Immediate	SVA	on	0	1	-	-	-	
/top/dut/empty_assert	Immediate	SVA	on	0	1	-	-	-	
/top/dut/underflow_assert	Immediate	SVA	on	0	1	-	-	-	
/top/dut/overflow_assert	Immediate	SVA	on	0	1	-	-	-	
/top/dut/almostfull_assert	Immediate	SVA	on	0	1	-	-	-	
/top/dut/almostempty_assert	Immediate	SVA	on	0	1	-	-	-	
/top/dut/wr_ack_assert	Immediate	SVA	on	0	1	-	-	-	
/top/TEST/#ublk#190244468#16/immed__17	Immediate	SVA	on	0	1	-	-	-	

Cover assertions:

Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Mer
/top/dut/wr_count_cover	SVA	✓	Off	3559	1	Unlimited	1	100%		✓	0	0	
/top/dut/rd_count_cover	SVA	✓	Off	614	1	Unlimited	1	100%		✓	0	0	
/top/dut/rd_wr_count_cover	SVA	✓	Off	1288	1	Unlimited	1	100%		✓	0	0	
/top/dut/rd_wr_count_empty_cover	SVA	✓	Off	268	1	Unlimited	1	100%		✓	0	0	
/top/dut/rd_wr_count_full_cover	SVA	✓	Off	205	1	Unlimited	1	100%		✓	0	0	
/top/dut/wr_ptr_cover	SVA	✓	Off	5115	1	Unlimited	1	100%		✓	0	0	
/top/dut/rd_ptr_cover	SVA	✓	Off	2107	1	Unlimited	1	100%		✓	0	0	
/top/dut/rst_count_cover	SVA	✓	Off	953	1	Unlimited	1	100%		✓	0	0	
/top/dut/rst_wr_ptr_cover	SVA	✓	Off	953	1	Unlimited	1	100%		✓	0	0	
/top/dut/rst_rd_ptr_cover	SVA	✓	Off	953	1	Unlimited	1	100%		✓	0	0	
/top/dut/full_cover	SVA	✓	Off	993	1	Unlimited	1	100%		✓	0	0	
/top/dut/empty_cover	SVA	✓	Off	2396	1	Unlimited	1	100%		✓	0	0	
/top/dut/underflow_cover	SVA	✓	Off	828	1	Unlimited	1	100%		✓	0	0	
/top/dut/overflow_cover	SVA	✓	Off	742	1	Unlimited	1	100%		✓	0	0	
/top/dut/almostfull_cover	SVA	✓	Off	1466	1	Unlimited	1	100%		✓	0	0	
/top/dut/almostempty_cover	SVA	✓	Off	2584	1	Unlimited	1	100%		✓	0	0	
/top/dut/wr_ack_cover	SVA	✓	Off	9872	1	Unlimited	1	100%		✓	0	0	

Functional Coverage:

Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/FIFO_coverage_pkg/FIFO_coverage		100.00%							
TYPE cg_FIFO		100.00%	100	100.00...		✓	auto(1)		
CVP cg_FIFO::cp_wr_en		100.00%	100	100.00...		✓			
CVP cg_FIFO::cp_rd_en		100.00%	100	100.00...		✓			
CVP cg_FIFO::cp_wr_ack		100.00%	100	100.00...		✓			
CVP cg_FIFO::cp_overflow		100.00%	100	100.00...		✓			
CVP cg_FIFO::cp_full		100.00%	100	100.00...		✓			
CVP cg_FIFO::cp_empty		100.00%	100	100.00...		✓			
CVP cg_FIFO::cp_almostfull		100.00%	100	100.00...		✓			
CVP cg_FIFO::cp_almostempty		100.00%	100	100.00...		✓			
CVP cg_FIFO::cp_underflow		100.00%	100	100.00...		✓			
CROSS cg_FIFO::cx_wr_ack		100.00%	100	100.00...		✓			
CROSS cg_FIFO::cx_overflow		100.00%	100	100.00...		✓			
CROSS cg_FIFO::cx_full		100.00%	100	100.00...		✓			
CROSS cg_FIFO::cx_empty		100.00%	100	100.00...		✓			
CROSS cg_FIFO::cx_almostfull		100.00%	100	100.00...		✓			
CROSS cg_FIFO::cx_almostempty		100.00%	100	100.00...		✓			
CROSS cg_FIFO::cx_underflow		100.00%	100	100.00...		✓			

Statement Coverage:

Code Coverage Analysis

Statements - by instance (/top/dut) | Statement

FIFO.sv

```
13 always @(posedge FIFOit.clk or negedge FIFOit.rst_n) begin
15 wr_ptr <= 0;
18 mem[wr_ptr] <= FIFOit.data_in;
20 wr_ptr <= wr_ptr + 1;
31 always @(posedge FIFOit.clk or negedge FIFOit.rst_n) begin
33 rd_ptr <= 0;
36 FIFOit.data_out <= mem[rd_ptr];
37 rd_ptr <= rd_ptr + 1;
40 always @(posedge FIFOit.clk or negedge FIFOit.rst_n) begin
42 count <= 0;
47 count <= count + 1;
50 count <= count - 1;
53 assign FIFOit.full = (count == FIFOit.FIFO_DEPTH)? 1 : 0;
54 assign FIFOit.empty = (count == 0)? 1 : 0;
55 assign FIFOit.underflow = (FIFOit.empty && FIFOit.rd_en)? 1 : 0;
57 assign FIFOit.almostempty = (count == 1)? 1 : 0;
59 assign FIFOit.wr_ack = ((count != FIFOit.FIFO_DEPTH) && FIFOit.wr_en)? 1 : 0;
60 assign FIFOit.almostfull = (count == FIFOit.FIFO_DEPTH - 1)? 1 : 0;
61 assign FIFOit.overflow = (FIFOit.full && FIFOit.wr_en)? 1 : 0;
```

Branch Coverage:

Code Coverage Analysis

Branches - by instance (/top/dut) | Branch

FIFO.sv

```
14 if (!FIFOit.rst_n) begin
17 else if (FIFOit.wr_en && count < FIFOit.FIFO_DEPTH) begin
32 if (!FIFOit.rst_n) begin
35 else if (FIFOit.rd_en && count != 0) begin
41 if (!FIFOit.rst_n) begin
44 else begin
45 if       ((({FIFOit.wr_en, FIFOit.rd_en} == 2'b10) && !FIFOit.full) ||
48 else if ((({FIFOit.wr_en, FIFOit.rd_en} == 2'b01) && !FIFOit.empty) ||
53 assign FIFOit.full = (count == FIFOit.FIFO_DEPTH)? 1 : 0;
54 assign FIFOit.empty = (count == 0)? 1 : 0;
55 assign FIFOit.underflow = (FIFOit.empty && FIFOit.rd_en)? 1 : 0;
57 assign FIFOit.almostempty = (count == 1)? 1 : 0;
59 assign FIFOit.wr_ack = ((count != FIFOit.FIFO_DEPTH) && FIFOit.wr_en)? 1 : 0;
60 assign FIFOit.almostfull = (count == FIFOit.FIFO_DEPTH - 1)? 1 : 0;
61 assign FIFOit.overflow = (FIFOit.full && FIFOit.wr_en)? 1 : 0;
```

Toggle Coverage:

Code Coverage Analysis

Toggles - by instance (/top/FIFOit) | Toggle

sim:/top/FIFOit

- almostempty
- almostfull
- clk
- data_in
- data_out
- empty
- full
- overflow
- rd_en
- rst_n
- underflow
- wr_ack
- wr_en