

FIFO Verification using SystemVerilog



Design file:

```
module fifo #(parameter FIFO_WIDTH = 16, FIFO_DEPTH = 8) (fifo_if.DUT fifoif);

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

localparam max_fifo_addr = $clog2(FIFO_DEPTH);
reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [max_fifo_addr:0] count;

always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
    if (!fifoif.rst_n) begin
        wr_ptr <= 0;
        fifoif.wr_ack <= 0; // wr_ack flag should be cleared on reset
        fifoif.overflow <= 0; // overflow flag should be cleared on reset
    end
    else if (fifoif.wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= fifoif.data_in;
        fifoif.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    end
    else begin
        fifoif.wr_ack <= 0;
        if (fifoif.full & fifoif.wr_en)
            fifoif.overflow <= 1;
        else
            fifoif.overflow <= 0;
    end
end

always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
    if (!fifoif.rst_n) begin
        rd_ptr <= 0;
        fifoif.underflow <= 0; // underflow flag should be cleared on reset
    end
    else if (fifoif.rd_en && count != 0) begin
        fifoif.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
    else begin // underflow flag logic should be sequential
        if (fifoif.empty & fifoif.rd_en) //
            fifoif.underflow <= 1; //
        else //
            fifoif.underflow <= 0; //
    end //
end

always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
    if (!fifoif.rst_n) begin
```

```

        count <= 0;
    end
    else begin
        if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b10) && !fifoif.full)
            count <= count + 1;
        else if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b01) && !fifoif.empty)
            count <= count - 1;
        else if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b11) && fifoif.empty) //
unhandled case when read/write and fifo is empty
            count <= count + 1;
        else if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b11) && fifoif.full) //
unhandled case when read/write and fifo is full
            count <= count - 1;
        end
    end
end

assign fifoif.full = (count == FIFO_DEPTH)? 1 : 0;
assign fifoif.empty = (count == 0)? 1 : 0;
// assign fifoif.underflow = (fifoif.empty && fifoif.rd_en)? 1 : 0; //
underflow flag logic should be sequential
assign fifoif.almostfull = (count == FIFO_DEPTH-1)? 1 : 0; // fifo is
almostfull when only one location left
assign fifoif.almostempty = (count == 1)? 1 : 0;

endmodule

```

Verification plan:

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
FIF01	When reset is asserted, pointers and flags must return to default values.	Directed at first of simulation then randomized.	-	reset assertion + scoreboard clearing state.
FIF02	When wr_en = 1 and FIFO not full, data_in should be written, wr_ack = 1, wr_ptr increments.	Randomly enable wr_en with varying data until FIFO fills.	wr_en_cp, wr_ack_cp, full_cp. Cross wr_en_cp x wr_ack_cp.	wr_ack assertion + scoreboard checks with wr_ack_ref.
FIF03	When rd_en = 1 and FIFO not empty, data should be read, data_out valid, rd_ptr increments.	Randomly enable rd_en under different fill-levels.	rd_en_cp, empty_cp. Cross rd_en_cp x empty_cp.	Scoreboard checks with data_out_ref.
FIF04	If wr_en = 1 and FIFO full, no new data stored, overflow = 1.	Force writes until FIFO full, then keep writing.	overflow_cp, full_cp. Cross wr_en_cp x full_cp x overflow_cp.	overflow assertion + scoreboard checks with overflow_ref.
FIF05	If rd_en = 1 and FIFO empty, underflow = 1, no valid data_out.	Force reads when FIFO empty.	underflow_cp, empty_cp. Cross rd_en_cp x empty_cp x underflow_cp.	underflow assertion + scoreboard checks with underflow_ref.
FIF07	Full = 1 when count = FIFO_DEPTH.	Fill FIFO until depth reached.	full_cp.	full assertion, scoreboard checks with full_ref.
FIF08	Empty = 1 when count=0.	Drain FIFO completely.	empty_cp.	empty assertion, scoreboard checks with empty_ref.
FIF09	Almostfull = 1 when count = FIFO_DEPTH-1.	Fill until one slot left.	almostfull_cp.	almostfull assertion, scoreboard checks with almostfull_ref.

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
FIF010	Almostempty = 1 when count = 1.	Drain until one entry left.	almostempty_cp.	almostempty assertion, scoreboard checks with almostempty_ref.
FIF011	When wr_ptr reaches FIFO_DEPTH-1 and another valid write occurs, pointer wraps to 0.	Fill FIFO past boundary (circular check).	-	wr_ptr_wraparound assertion.
FIF012	When rd_ptr reaches FIFO_DEPTH-1 and another valid read occurs, pointer wraps to 0.	Drain FIFO past boundary (circular check).	-	rd_ptr_wraparound assertion.
FIF013	Count should never exceed FIFO_DEPTH or go negative.	Random sequences of writes/reads.	-	ptr_threshold assertion.

Interface:

```
interface fifo_if #(parameter FIFO_WIDTH = 16, FIFO_DEPTH = 8) (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 wr_ack;
    logic full;
    logic empty;
    logic almostfull;
    logic almostempty;
    logic overflow;
    logic underflow;

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

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

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

Top file:

```
module fifo_top();

    bit clk;
    initial begin
        clk = 0;
        forever begin
            #10 clk = ~clk;
        end
    end

    fifo_if fifoif (clk);
    fifo DUT (fifoif);
    fifo_tb TEST (fifoif);
    fifo_monitor MONITOR (fifoif);

`ifdef SIM
    always_comb begin
        if (!fifoif.rst_n) begin
            assert final ((fifoif.empty == 1) && (fifoif.full == 0) &&
                (fifoif.almostfull == 0) && (fifoif.almostempty == 0) &&
                (fifoif.overflow == 0) && (fifoif.underflow == 0) &&
                (fifoif.wr_ack == 0));
        end
    end
`endif

endmodule
```

Testbench file:

```
import fifo_shared_pkg::*;
import fifo_transaction_pkg::*;

module fifo_tb(fifo_if.TEST fifoif);

    fifo_transaction F_txn;

    initial begin
        $display("Starting FIFO testbench...");
        test_finished = 0;
        F_txn = new();

        $display("Reset Asserted...");
        reset_sequence(F_txn);
        $display("Reset Deasserted...");

        $display("Starting write only sequence...");
        write_only_sequence(F_txn);

        $display("Starting read only sequence...");
        read_only_sequence(F_txn);

        $display("Starting read/write sequence...");
        read_write_sequence(F_txn);
        test_finished = 1;
    end

    task drive_inputs(fifo_transaction F_txn);
        fifoif.rst_n = F_txn.rst_n;
        fifoif.wr_en = F_txn.wr_en;
        fifoif.rd_en = F_txn.rd_en;
        fifoif.data_in = F_txn.data_in;
        ->> sample_event;
        @(negedge fifoif.clk);
    endtask

    task reset_sequence(fifo_transaction F_txn);
        F_txn.rst_n = 0;
        F_txn.wr_en = 0;
        F_txn.rd_en = 0;
        F_txn.data_in = 0;
        drive_inputs(F_txn);
    endtask

    task write_only_sequence(fifo_transaction F_txn);
        F_txn = new(.rd_dist(30), .wr_dist(70));
        repeat(NUM_TEST_CASES) begin
```



```

        assert(F_txn.randomize());
        drive_inputs(F_txn);
    end
endtask

task read_only_sequence(fifo_transaction F_txn);
    F_txn = new(.rd_dist(70), .wr_dist(30));
    repeat(NUM_TEST_CASES) begin
        assert(F_txn.randomize());
        drive_inputs(F_txn);
    end
endtask

task read_write_sequence(fifo_transaction F_txn);
    F_txn = new(.rd_dist(50), .wr_dist(50));
    repeat(NUM_TEST_CASES) begin
        assert(F_txn.randomize());
        drive_inputs(F_txn);
    end
endtask

endmodule

```

Monitor file:

```
import fifo_shared_pkg::*;
import fifo_transaction_pkg::*;
import fifo_coverage_pkg::*;
import fifo_scoreboard_pkg::*;

module fifo_monitor(fifo_if.MONITOR fifoif);
    fifo_transaction F_txn;
    fifo_coverage F_cvr;
    fifo_scoreboard F_sb;

    initial begin
        F_txn = new();
        F_cvr = new();
        F_sb = new();

        forever begin
            wait(sample_event);
            @(negedge fifoif.clk);
            F_txn.rst_n = fifoif.rst_n;
            F_txn.wr_en = fifoif.wr_en;
            F_txn.rd_en = fifoif.rd_en;
            F_txn.data_in = fifoif.data_in;
            F_txn.data_out = fifoif.data_out;
            F_txn.wr_ack = fifoif.wr_ack;
            F_txn.full = fifoif.full;
            F_txn.empty = fifoif.empty;
            F_txn.almostfull = fifoif.almostfull;
            F_txn.almostempty = fifoif.almostempty;
            F_txn.overflow = fifoif.overflow;
            F_txn.underflow = fifoif.underflow;

            fork
                F_cvr.sample_data(F_txn);
                F_sb.check_data(F_txn);
            join

            #0;
            if (test_finished) begin
                $display("Simulation Completed: %0d test cases executed.",
(NUM_TEST_CASES * 3));
                $display("Test Summary: Passed = %0d, Failed = %0d",
correct_count, error_count);
                $stop;
            end
        end
    end
endmodule
```

Coverage collector package:

```
package fifo_coverage_pkg;
import fifo_transaction_pkg::*;

class fifo_coverage;
    fifo_transaction F_cvg_txn = new();

    covergroup cvr_grp;
        wr_en_cp: coverpoint F_cvg_txn.wr_en{
            bins wr_en_0 = {0};
            bins wr_en_1 = {1};
            option.weight = 0;
        }
        rd_en_cp: coverpoint F_cvg_txn.rd_en{
            bins rd_en_0 = {0};
            bins rd_en_1 = {1};
            option.weight = 0;
        }
        wr_ack_cp: coverpoint F_cvg_txn.wr_ack{
            bins wr_ack_0 = {0};
            bins wr_ack_1 = {1};
            option.weight = 0;
        }
        overflow_cp: coverpoint F_cvg_txn.overflow{
            bins overflow_0 = {0};
            bins overflow_1 = {1};
            option.weight = 0;
        }
        full_cp: coverpoint F_cvg_txn.full{
            bins full_0 = {0};
            bins full_1 = {1};
            option.weight = 0;
        }
        empty_cp: coverpoint F_cvg_txn.empty{
            bins empty_0 = {0};
            bins empty_1 = {1};
            option.weight = 0;
        }
        almostfull_cp: coverpoint F_cvg_txn.almostfull{
            bins almostfull_0 = {0};
            bins almostfull_1 = {1};
            option.weight = 0;
        }
        almostempty_cp: coverpoint F_cvg_txn.almostempty{
            bins almostempty_0 = {0};
            bins almostempty_1 = {1};
            option.weight = 0;
        }
    endcovergroup
endclass
```

```

underflow_cp: coverpoint F_cvg_txn.underflow{
    bins underflow_0 = {0};
    bins underflow_1 = {1};
    option.weight = 0;
}

wr_ack_cross: cross wr_en_cp, rd_en_cp, wr_ack_cp{
    illegal_bins wr_en_0 = binsof(wr_ack_cp) intersect {1} &&
binsof(wr_en_cp) intersect {0};
}

full_cross: cross wr_en_cp, rd_en_cp, full_cp{
    illegal_bins rd_en = binsof(full_cp) intersect {1} &&
binsof(rd_en_cp) intersect {1} ;
}

almostfull_cross: cross wr_en_cp, rd_en_cp, almostfull_cp;

overflow_cross: cross wr_en_cp, rd_en_cp, overflow_cp{
    illegal_bins wr_en_0 = binsof(overflow_cp) intersect {1} &&
binsof(wr_en_cp) intersect {0};
}

empty_cross: cross wr_en_cp, rd_en_cp, empty_cp;

almostempty_cross: cross wr_en_cp, rd_en_cp, almostempty_cp;

underflow_cross: cross wr_en_cp, rd_en_cp, underflow_cp{
    illegal_bins rd_en_1 =binsof(underflow_cp) intersect {1} &&
binsof(rd_en_cp) intersect {0};
}
endgroup

function new();
    cvr_grp = new();
endfunction

function void sample_data(fifo_transaction F_txn);
    F_cvg_txn = F_txn;
    cvr_grp.sample();
endfunction
endclass : fifo_coverage

endpackage : fifo_coverage_pkg

```

Scoreboard package:

```
package fifo_scoreboard_pkg;
import fifo_shared_pkg::*;
import fifo_transaction_pkg::*;

class fifo_scoreboard #(parameter FIFO_WIDTH = 16, FIFO_DEPTH = 8);
    logic [FIFO_WIDTH-1:0] data_out_ref;
    logic wr_ack_ref;
    logic full_ref;
    logic empty_ref;
    logic almostfull_ref;
    logic almostempty_ref;
    logic overflow_ref;
    logic underflow_ref;

    bit [FIFO_WIDTH-1:0] fifo_ref [$];

    function void check_data(fifo_transaction F_txn);
        reference_model(F_txn);

        if ((data_out_ref != F_txn.data_out) || (wr_ack_ref != F_txn.wr_ack)
            || (full_ref != F_txn.full) ||
                (empty_ref != F_txn.empty) || (almostfull_ref != F_txn.almostfull) ||
                (almostempty_ref != F_txn.almostempty) ||
                (overflow_ref != F_txn.overflow) || (underflow_ref != F_txn.underflow)) begin

            $display("time: %0t Comparsion failed, Transaction received from
DUT: %s While the reference data_out_ref = 0x%0h, wr_ack_ref = %0b, full_ref =
%0b, empty_ref = %0b, almostfull_ref = %0b, almostempty_ref = %0b,
overflow_ref = %0b, underflow_ref = %0b",
                $time, F_txn.convert2string(), data_out_ref, wr_ack_ref,
full_ref, empty_ref, almostfull_ref, almostempty_ref, overflow_ref,
underflow_ref);

            error_count++;
        end else begin
            $display("time: %0t Comparsion succeeded, Transaction received
from DUT: %s ", $time, F_txn.convert2string());

            correct_count++;
        end
    endfunction

    function void reference_model(fifo_transaction F_txn);
        if (!F_txn.rst_n) begin
            fifo_ref.delete();
        end
    end
endclass
```

```

        wr_ack_ref = 0;
        overflow_ref = 0;
        underflow_ref = 0;
    end else begin
        if (F_txn.rd_en) begin
            if (!empty_ref) begin
                data_out_ref = fifo_ref.pop_front();
            end else begin
                underflow_ref = 1;
            end
        end else begin
            underflow_ref = 0;
        end

        if (F_txn.wr_en) begin
            if (!full_ref) begin
                fifo_ref.push_back(F_txn.data_in);
                wr_ack_ref = 1;
            end else begin
                wr_ack_ref = 0;
                overflow_ref = 1;
            end
        end else begin
            wr_ack_ref = 0;
            overflow_ref = 0;
        end
    end

    full_ref = (fifo_ref.size() == FIFO_DEPTH);
    empty_ref = (fifo_ref.size() == 0);
    almostfull_ref = (fifo_ref.size() == FIFO_DEPTH - 1);
    almostempty_ref = (fifo_ref.size() == 1);
endfunction
endclass : fifo_scoreboard

endpackage : fifo_scoreboard_pkg

```

Transaction package:

```
package fifo_transaction_pkg;

class fifo_transaction #(parameter FIFO_WIDTH = 16, FIFO_DEPTH = 8);
    rand bit rst_n;
    rand bit wr_en;
    rand bit rd_en;
    rand bit [FIFO_WIDTH-1:0] data_in;
    logic [FIFO_WIDTH-1:0] data_out;
    logic wr_ack;
    logic full;
    logic empty;
    logic almostfull;
    logic almostempty;
    logic overflow;
    logic underflow;

    int RD_EN_ON_DIST, WR_EN_ON_DIST;

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

    constraint c_rst_n {
        rst_n dist {1 := 95, 0 := 5};
    }

    constraint c_wr_en {
        wr_en dist {1 := WR_EN_ON_DIST, 0 := (100 - WR_EN_ON_DIST)};
    }

    constraint c_rd_en {
        rd_en dist {1 := RD_EN_ON_DIST, 0 := (100 - RD_EN_ON_DIST)};
    }

    function string convert2string();
        return $sformatf("rst_n = %0b, wr_en = %0b, rd_en = %0b, data_in = %x%0h, data_out = %x%0h, wr_ack = %0b, full = %0b, empty = %0b, almostfull = %0b, almostempty = %0b, overflow = %0b, underflow = %0b",
            rst_n, wr_en, rd_en, data_in, data_out, wr_ack, full, empty, almostfull, almostempty, overflow, underflow);
    endfunction
endclass : fifo_transaction

endpackage : fifo_transaction_pkg
```

Shared package:

```
package fifo_shared_pkg;  
  int NUM_TEST_CASES = 100;  
  
  int error_count = 0;  
  int correct_count = 0;  
  bit test_finished = 0;  
  
  event sample_event;  
endpackage : fifo_shared_pkg
```


Assertions:

```
`ifdef SIM
    always_comb begin
        if (!fifoif.rst_n) begin
            assert_rst_n: assert final(wr_ptr == {max_fifo_addr{1'b0}} &&
rd_ptr == {max_fifo_addr{1'b0}} && count == {max_fifo_addr + 1{1'b0}});
            end
        end

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

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

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

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

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

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

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

    property wr_ptr_wraparound_p;
        @(posedge fifoif.clk) disable iff (!fifoif.rst_n)
            (wr_ptr == (FIFO_DEPTH - 1) && fifoif.wr_en && !fifoif.full) |=>
(wr_ptr == {max_fifo_addr{1'b0}});
`endif
```

```

endproperty

property rd_ptr_wraparound_p;
    @(posedge fifoif.clk) disable iff (!fifoif.rst_n)
        (rd_ptr == FIFO_DEPTH-1 && fifoif.rd_en && !fifoif.empty) | =>
(rd_ptr == {max_fifo_addr{1'b0}});
endproperty

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

assert_wr_ack: assert property(wr_ack_p);
assert_overflow: assert property(overflow_p);
assert_underflow: assert property(underflow_p);
assert_empty: assert property(empty_p);
assert_full: assert property(full_p);
assert_almostfull: assert property(almostfull_p);
assert_almostempty: assert property(almostempty_p);
assert_wr_ptr_wraparound: assert property(wr_ptr_wraparound_p);
assert_rd_ptr_wraparound: assert property(rd_ptr_wraparound_p);
assert_ptr_threshold: assert property(ptr_threshold_p);

cover_wr_ack: cover property(wr_ack_p);
cover_overflow: cover property(overflow_p);
cover_underflow: cover property(underflow_p);
cover_empty: cover property(empty_p);
cover_full: cover property(full_p);
cover_almostfull: cover property(almostfull_p);
cover_almostempty: cover property(almostempty_p);
cover_wr_ptr_wraparound: cover property(wr_ptr_wraparound_p);
cover_rd_ptr_wraparound: cover property(rd_ptr_wraparound_p);
cover_ptr_threshold: cover property(ptr_threshold_p);

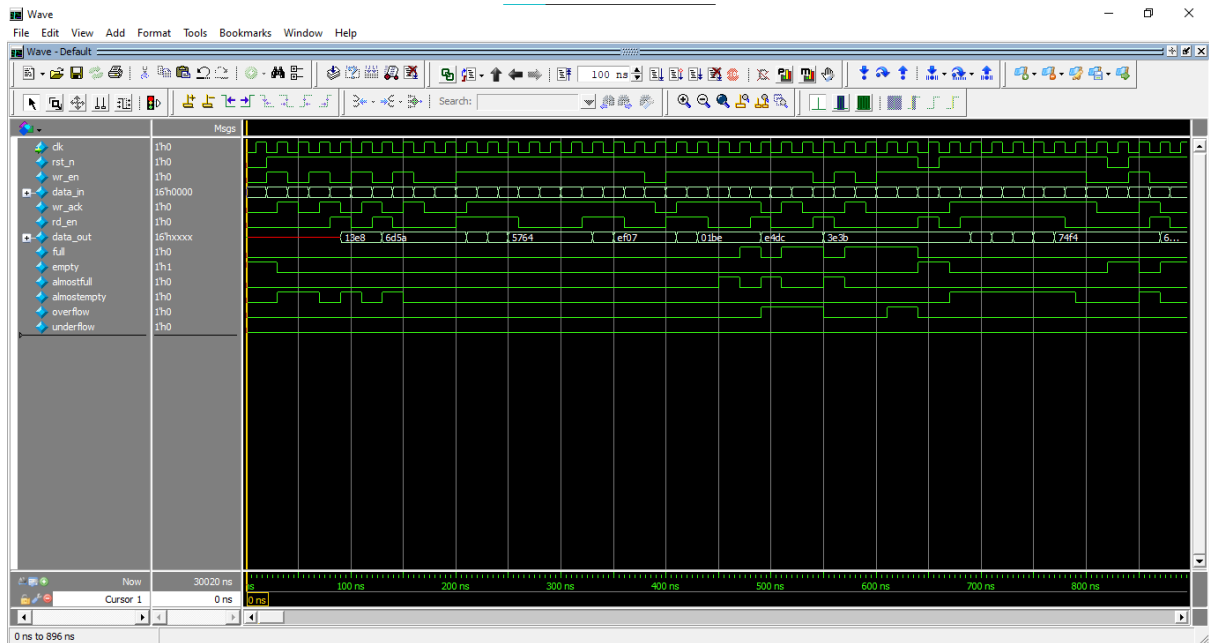
`endif

```

Do file:

```
vlib work
vlog -f src_files.list +cover -covercells +define+SIM
vsim -voptargs=+acc work.fifo_top -cover -classdebug
add wave /fifo_top/fifoif/*
coverage save FIFO_tb.ucdb -onexit
run -all
quit -sim
vcover report FIFO_tb.ucdb -details -annotate -all -output FIFO_coverage_rpt.txt
```

Simulation snippet:



Transcript:

```
# Starting FIFO testbench...

# Reset Asserted...

# time: 20 Comparsion succeeded, Transaction received from DUT: rst_n = 0,
wr_en = 0, rd_en = 0, data_in = 0x0, data_out = 0xx, wr_ack = 0, full = 0,
empty = 1, almostfull = 0, almostempty = 0, overflow = 0, underflow = 0

# Reset Deasserted...

# Starting write only sequence...

# time: 40 Comparsion succeeded, Transaction received from DUT: rst_n = 1,
wr_en = 1, rd_en = 0, data_in = 0x13e8, data_out = 0xx, wr_ack = 1, full = 0,
empty = 0, almostfull = 0, almostempty = 1, overflow = 0, underflow = 0

...

# Starting read only sequence...
# time: 10040 Comparsion succeeded, Transaction received from DUT: rst_n = 1,
wr_en = 0, rd_en = 1, data_in = 0xfa95, data_out = 0xac6c, wr_ack = 0, full =
0, empty = 0, almostfull = 0, almostempty = 0, overflow = 0, underflow = 0
...
# Starting read/write sequence...
# time: 20040 Comparsion succeeded, Transaction received from DUT: rst_n = 1,
wr_en = 0, rd_en = 1, data_in = 0x1351, data_out = 0x77a9, wr_ack = 0, full =
0, empty = 1, almostfull = 0, almostempty = 0, overflow = 0, underflow = 1
...

# Simulation Completed: 1500 test cases executed.

# Test Summary: Passed = 1501, Failed = 0

# ** Note: $stop      : FIFO_monitor.sv(43)
```

Bugs report:

Issue: overflow and wr_ack signals were not properly reset during initialization.

```
always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
    if (!fifoif.rst_n) begin
        ...
        fifoif.wr_ack <= 0;
        fifoif.overflow <= 0;
    end
    ...
end
```

Issue: underflow should be implemented as a sequential signal, earlier versions treated it as a combinational output.

```
always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
    if (!fifoif.rst_n) begin
        ...
        fifoif.underflow <= 0;
    end
    ...
    else begin
        if (fifoif.empty & fifoif.rd_en)
            fifoif.underflow <= 1;
        else
            fifoif.underflow <= 0;
        end
    end
end
```

Issue: When both rd_en and wr_en were asserted and the FIFO was empty, only the writing will take place (read ignored) ,and if the FIFO was full, only the reading will take place (write ignored).

While these cases were partially handled in the control logic, the counter update logic did not account for them, resulting in incorrect count values.

```
always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
    ...
    else if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b11) && fifoif.empty)
        count <= count + 1;
    else if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b11) && fifoif.full)
        count <= count - 1;
    end
end
```

Code coverage:

```
> vcover report FIFO_tb.ucdb -details -annotate -all -output coverage_rpt.txt
```

```
=====
Branch Coverage:
```

Enabled Coverage	Bins	Hits	Misses	Coverage
-----	----	----	-----	-----
Branches	27	27	0	100.00%

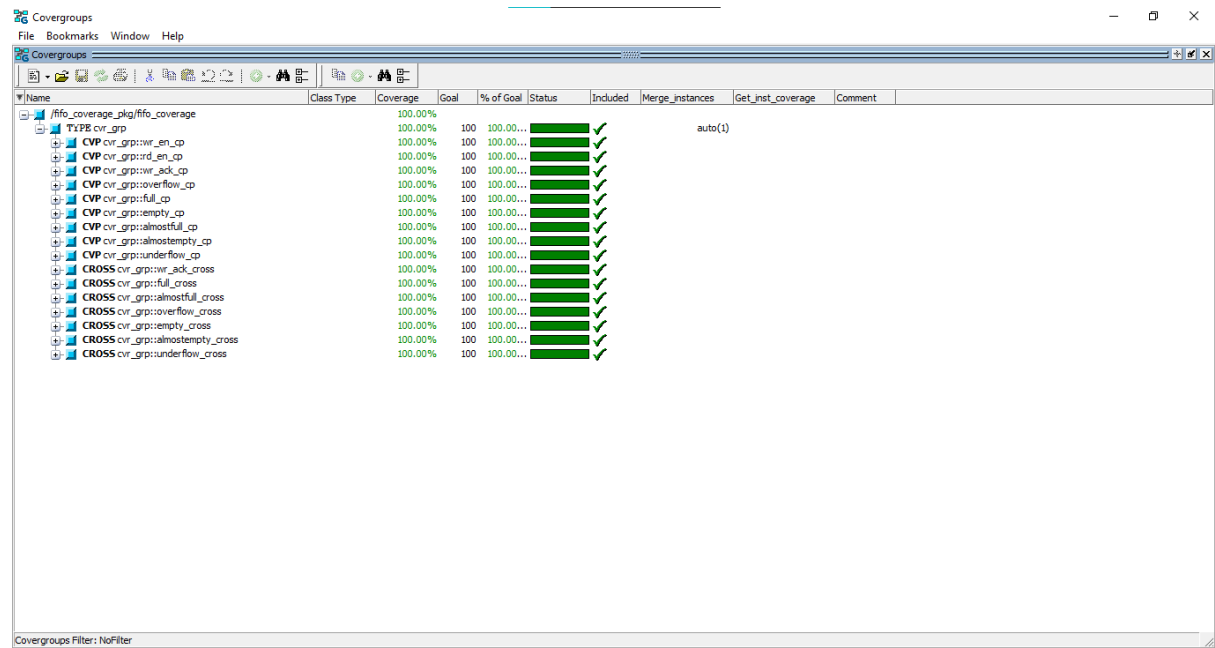
```
=====
Statement Coverage:
```

Enabled Coverage	Bins	Hits	Misses	Coverage
-----	----	----	-----	-----
Statements	28	28	0	100.00%

```
=====
Toggle Coverage:
```

Enabled Coverage	Bins	Hits	Misses	Coverage
-----	----	----	-----	-----
Toggles	86	86	0	100.00%

Functional coverage:



The screenshot shows the Covergroups tool interface. The main table lists various coverage groups, all of which have achieved 100.00% coverage. The groups are organized into categories like T1P8, CVP, and CROSS. Each row includes a checkbox, the group name, its class type, coverage percentage, goal, percentage of goal, status, and a visual progress bar. A comment 'auto(1)' is visible for the first group.

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/fifo_coverage_pkg/fifo_coverage		100.00%	100	100.00...					
[-] T1P8 cvr_grp		100.00%	100	100.00...					auto(1)
[-] CVP cvr_grp:wr_en_cp		100.00%	100	100.00...					
[-] CVP cvr_grp:rd_en_cp		100.00%	100	100.00...					
[-] CVP cvr_grp:wr_ack_cp		100.00%	100	100.00...					
[-] CVP cvr_grp:overflow_cp		100.00%	100	100.00...					
[-] CVP cvr_grp:full_cp		100.00%	100	100.00...					
[-] CVP cvr_grp:empty_cp		100.00%	100	100.00...					
[-] CVP cvr_grp:almostfull_cp		100.00%	100	100.00...					
[-] CVP cvr_grp:almostempty_cp		100.00%	100	100.00...					
[-] CVP cvr_grp:underflow_cp		100.00%	100	100.00...					
[-] CROSS cvr_grp:wr_ack_cross		100.00%	100	100.00...					
[-] CROSS cvr_grp:full_cross		100.00%	100	100.00...					
[-] CROSS cvr_grp:almostfull_cross		100.00%	100	100.00...					
[-] CROSS cvr_grp:overflow_cross		100.00%	100	100.00...					
[-] CROSS cvr_grp:empty_cross		100.00%	100	100.00...					
[-] CROSS cvr_grp:almostempty_cross		100.00%	100	100.00...					
[-] CROSS cvr_grp:underflow_cross		100.00%	100	100.00...					

Covergroups Filter: NoFilter

The screenshot displays the 'Assertions' window of a debugger. The window has a menu bar (File, Edit, View, Add, Bookmarks, Window, Help) and a toolbar with various icons. Below the toolbar is a table listing 14 assertions. The table columns are: Name, Assertion Type, Language, Enable, Failure Count, Pass Count, Active Count, Memory, Peak Memory, Peak Memory Time, Cumulative Threads, ATV, Assertion Expression, and Include. The assertions are listed with their names, types (Immediate, Concurrent, or Immediate), languages (SVA), and enable status (on). The failure and pass counts are mostly 0, and the active counts are 1. The memory and peak memory values are mostly 0B. The peak memory time values are mostly 0 ns. The cumulative threads and ATV values are mostly 0. The assertion expressions are listed in the 'Assertion Expression' column, and the 'Include' column shows a green checkmark for all assertions.

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Include
/fifo_top/DUT/assert_stat_n	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert (wr_ptr == {3'0})...	✓
/fifo_top/DUT/assert_wr_ack	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@posedge ffrif...	✓
/fifo_top/DUT/assert_overflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@posedge ffrif...	✓
/fifo_top/DUT/assert_underflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@posedge ffrif...	✓
/fifo_top/DUT/assert_empty	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@posedge ffrif...	✓
/fifo_top/DUT/assert_full	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@posedge ffrif...	✓
/fifo_top/DUT/assert_almostfull	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@posedge ffrif...	✓
/fifo_top/DUT/assert_almostempty	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@posedge ffrif...	✓
/fifo_top/DUT/assert_wr_ptr_around	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@posedge ffrif...	✓
/fifo_top/DUT/assert_rd_ptr_around	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@posedge ffrif...	✓
/fifo_top/DUT/assert_ptr_threshold	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@posedge ffrif...	✓
/fifo_top/TEST/write_only_sequence/#ublk#217929410#47/Immed__48	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert (F_bmn.randomize()	✓
/fifo_top/TEST/read_only_sequence/#ublk#217929410#55/Immed__56	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert (F_bmn.randomize()	✓
/fifo_top/TEST/read_write_sequence/#ublk#217929410#63/Immed__64	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert (F_bmn.randomize()	✓
/fifo_top/#ublk#265645472#18/Immed__19	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert (ffifof.empty&-ffif...	✓

Assertions Filter: NoFilter