

Synchronous FIFO Verification using UVM & SystemVerilog



Author: Mohamed Atef Yousef

Design file:

```
import fifo_shared_pkg::*;

module fifo (fifo_if.DUT fifoif);

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 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

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

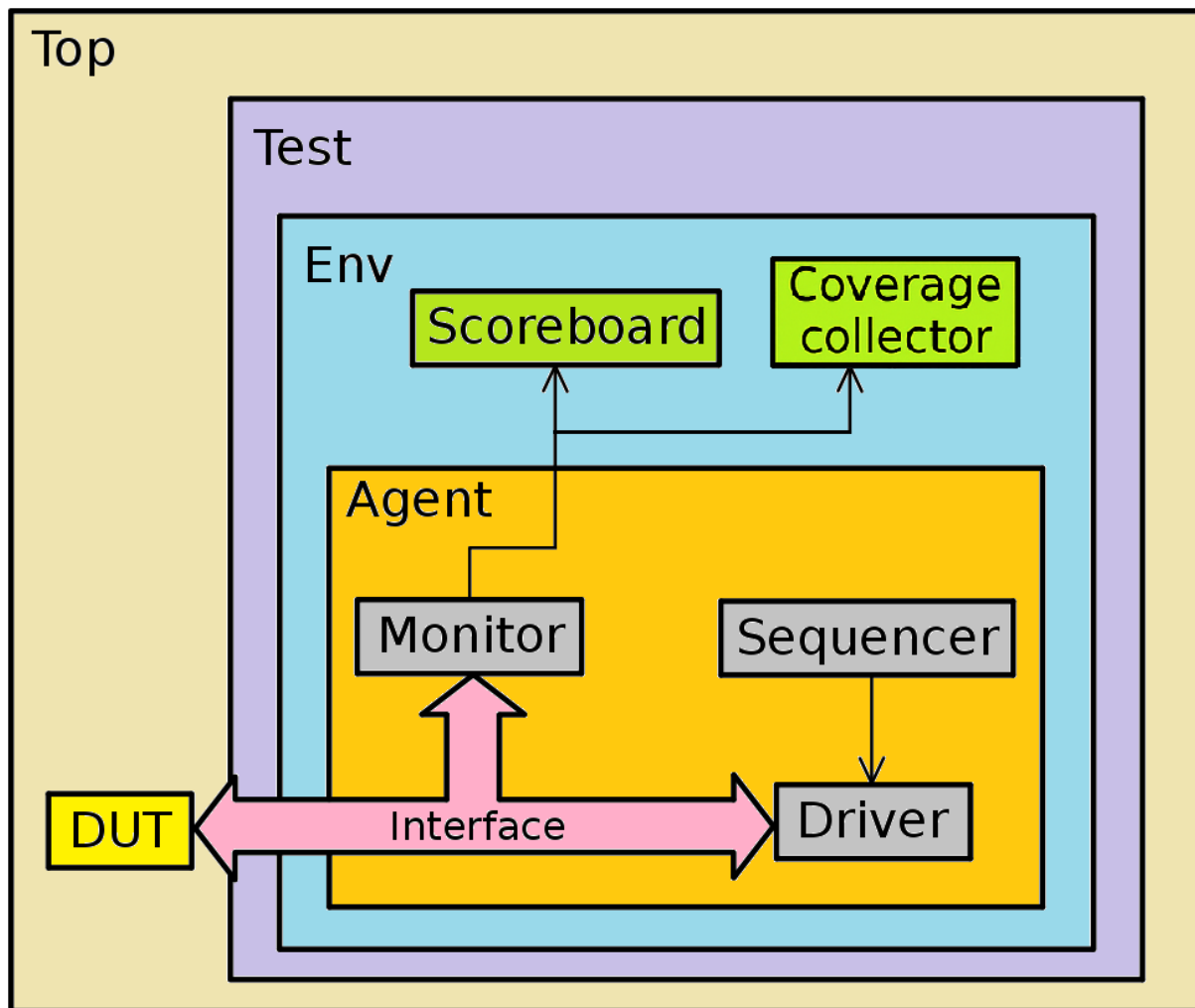
```

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.

UVM structure:



1. Top Module

The verification process begins in the **Top Module**, where key setup components are instantiated, this includes:

- Clock generation
- Interface instantiation and connection to the DUT
- Assertion binding for checks
- UVM test initiation through a call to `run_test()`

The interface handle is passed to the UVM environment using the **UVM configuration database** (`uvm_config_db`).

2. Test Layer

Within the **Test**, the **Environment (env)** is built and configured.

- The **virtual interface** is retrieved from the configuration database.
- The **configuration object** is also set into the database for retrieval by the **Agent**.
- During the **run phase**, the test triggers the appropriate **sequences**, which generate and send **transactions** to the **sequencer**.

In this verification plan, four main sequences are defined:

- **Reset sequence**
- **Write-only sequence**
- **Read-only sequence**
- **Read/Write sequence**

Each of these sequences is started from the **Test**, controlling the stimulus applied to the DUT.

3. Sequencer-Driver Interaction

The **sequencer** forwards the generated transactions to the **driver**. The **driver** then drives these transactions onto the DUT signals through the **virtual interface (fifo_if)**, ensuring proper timing.

4. Monitor and Analysis Flow

The **monitor** continuously observes DUT signals through the same **virtual interface**. It collects signal-level information, converts it into **sequence items**, and publishes

them through an **analysis port**. These analysis ports are connected to both the **scoreboard** and the **coverage collector**, enabling parallel checking and coverage sampling.

5. Scoreboard

The **scoreboard** receives sequence items via the analysis port and performs functional checking by:

- Feeding inputs to a **reference model**
- Comparing expected outputs against actual DUT outputs
- Reporting mismatches and tracking pass/fail statistics

6. Coverage Collector

The **coverage collector** also receives the sequence items from the monitor and samples relevant data fields to measure **functional coverage**. This helps ensure all scenarios and corner cases are exercised.

7. Agent and Environment Integration

- The **Agent** builds the **sequencer**, **driver**, and **monitor**.
- It retrieves the configuration object, connects its analysis port to the environment's, and links the driver's port to the sequencer's export.
- The **Environment** builds the **Agent**, **Scoreboard**, and **Coverage Collector**, connecting the respective analysis ports to facilitate data flow for checking and coverage collection.

Interface:

```
import fifo_shared_pkg::*;

interface fifo_if (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
    );
endinterface : fifo_if
```

Top file:

```
import uvm_pkg::*;
import fifo_test_pkg::*;
`include "uvm_macros.svh"

module fifo_top();

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

    fifo_if fifoif (clk);
    fifo DUT (fifoif);
    bind fifo fifo_sva ASSERT (fifoif);

    initial begin
        uvm_config_db #(virtual fifo_if)::set(null, "uvm_test_top",
"FIFO_VIF", fifoif);
        run_test("fifo_test");
    end

endmodule
```

Test package:

```
package fifo_test_pkg;
import uvm_pkg::*;
import fifo_env_pkg::*;
import fifo_config_obj_pkg::*;
import fifo_reset_seq_pkg::*;
import fifo_write_seq_pkg::*;
import fifo_read_seq_pkg::*;
import fifo_read_write_seq_pkg::*;
`include "uvm_macros.svh"

class fifo_test extends uvm_test;
    `uvm_component_utils(fifo_test)

    fifo_env env;
    fifo_config_obj fifo_cfg;
    fifo_reset_seq reset_seq;
    fifo_write_seq write_seq;
    fifo_read_seq read_seq;
    fifo_read_write_seq read_write_seq;

    function new(string name = "fifo_test", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        env = fifo_env::type_id::create("env", this);
        fifo_cfg = fifo_config_obj::type_id::create("fifo_cfg");
        reset_seq = fifo_reset_seq::type_id::create("reset_seq");
        write_seq = fifo_write_seq::type_id::create("write_seq");
        read_seq = fifo_read_seq::type_id::create("read_seq");
        read_write_seq
= fifo_read_write_seq::type_id::create("read_write_seq");

        if (!uvm_config_db #(virtual fifo_if)::get(this, "", "FIFO_VIF",
fifo_cfg.fifo_vif)) begin
            `uvm_fatal("BUILD_PHASE", "Test - Unable to get virtual interface
of the fifo")
        end

        uvm_config_db #(fifo_config_obj)::set(this, "*", "CFG", fifo_cfg);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        phase.raise_objection(this);
    endtask
endclass
```

```

// reset sequence
`uvm_info("RUN_PHASE", "Reset Asserted", UVM_LOW)
reset_seq.start(env.agent.sequencer);
`uvm_info("RUN_PHASE", "Reset Deasserted", UVM_LOW)

// write only sequence
`uvm_info("RUN_PHASE", "Write Only Sequence Started", UVM_LOW)
write_seq.start(env.agent.sequencer);
`uvm_info("RUN_PHASE", "Write Only Sequence Ended", UVM_LOW)

// read only sequence
`uvm_info("RUN_PHASE", "Read Only Sequence Started", UVM_LOW)
read_seq.start(env.agent.sequencer);
`uvm_info("RUN_PHASE", "Read Only Sequence Ended", UVM_LOW)

// read/write sequence
`uvm_info("RUN_PHASE", "Read/Write Sequence Started", UVM_LOW)
read_write_seq.start(env.agent.sequencer);
`uvm_info("RUN_PHASE", "Read/Write Sequence Ended", UVM_LOW)

phase.drop_objection(this);
endtask
endclass : fifo_test

endpackage : fifo_test_pkg

```

Env. package:

```
package fifo_env_pkg;
import uvm_pkg::*;
import fifo_agent_pkg::*;
import fifo_scoreboard_pkg::*;
import fifo_coverage_pkg::*;
`include "uvm_macros.svh"

class fifo_env extends uvm_env;
    `uvm_component_utils(fifo_env)

    fifo_agent agent;
    fifo_scoreboard scoreboard;
    fifo_coverage coverage;

    function new(string name = "fifo_env", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        agent = fifo_agent::type_id::create("agent", this);
        scoreboard = fifo_scoreboard::type_id::create("scoreboard", this);
        coverage = fifo_coverage::type_id::create("coverage", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        agent.analysis_port.connect(scoreboard.analysis_export);
        agent.analysis_port.connect(coverage.analysis_export);
    endfunction
endclass : fifo_env

endpackage : fifo_env_pkg
```

Agent package:

```
package fifo_agent_pkg;
import uvm_pkg::*;
import fifo_sequencer_pkg::*;
import fifo_seq_item_pkg::*;
import fifo_driver_pkg::*;
import fifo_monitor_pkg::*;
import fifo_config_obj_pkg::*;
`include "uvm_macros.svh"

class fifo_agent extends uvm_agent;
    `uvm_component_utils(fifo_agent)

    fifo_sequencer sequencer;
    fifo_driver driver;
    fifo_monitor monitor;
    fifo_config_obj fifo_cfg;
    uvm_analysis_port #(fifo_seq_item) analysis_port;

    function new(string name = "fifo_agent", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        if (!uvm_config_db #(fifo_config_obj)::get(this, "", "CFG", fifo_cfg))
            `uvm_fatal("BUILD_PHASE", "Agent - Unable to get the configuration
object")

        sequencer = fifo_sequencer::type_id::create("sequencer", this);
        driver = fifo_driver::type_id::create("driver", this);
        monitor = fifo_monitor::type_id::create("monitor", this);

        analysis_port = new("analysis_port", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        driver.fifo_vif = fifo_cfg.fifo_vif;
        monitor.fifo_vif = fifo_cfg.fifo_vif;

        driver.seq_item_port.connect(sequencer.seq_item_export);
        monitor.analysis_port.connect(analysis_port);
    endfunction
endclass : fifo_agent

endpackage : fifo_agent_pkg
```

Driver package:

```
package fifo_driver_pkg;
import uvm_pkg::*;
import fifo_seq_item_pkg::*;
`include "uvm_macros.svh"

class fifo_driver extends uvm_driver #(fifo_seq_item);
    `uvm_component_utils(fifo_driver)

    virtual fifo_if fifo_vif;
    fifo_seq_item seq_item;

    function new(string name = "fifo_driver", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            seq_item = fifo_seq_item::type_id::create("seq_item");
            seq_item_port.get_next_item(seq_item);
            fifo_vif.rst_n = seq_item.rst_n;
            fifo_vif.wr_en = seq_item.wr_en;
            fifo_vif.rd_en = seq_item.rd_en;
            fifo_vif.data_in = seq_item.data_in;
            @(negedge fifo_vif.clk);
            seq_item_port.item_done();
            `uvm_info("RUN_PHASE", seq_item.convert2string_stimulus(),
UVM_HIGH)
        end
    endtask
endclass : fifo_driver

endpackage : fifo_driver_pkg
```

Monitor package:

```
package fifo_monitor_pkg;
import uvm_pkg::*;
import fifo_seq_item_pkg::*;
`include "uvm_macros.svh"

class fifo_monitor extends uvm_monitor;
    `uvm_component_utils(fifo_monitor)

    virtual fifo_if fifo_vif;
    fifo_seq_item seq_item;
    uvm_analysis_port #(fifo_seq_item) analysis_port;

    function new(string name = "fifo_monitor", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        analysis_port = new("analysis_port", this);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            seq_item = fifo_seq_item::type_id::create("seq_item");
            @(negedge fifo_vif.clk);
            seq_item.rst_n = fifo_vif.rst_n;
            seq_item.wr_en = fifo_vif.wr_en;
            seq_item.rd_en = fifo_vif.rd_en;
            seq_item.data_in = fifo_vif.data_in;
            seq_item.data_out = fifo_vif.data_out;
            seq_item.wr_ack = fifo_vif.wr_ack;
            seq_item.full = fifo_vif.full;
            seq_item.empty = fifo_vif.empty;
            seq_item.almostfull = fifo_vif.almostfull;
            seq_item.almostempty = fifo_vif.almostempty;
            seq_item.underflow = fifo_vif.underflow;
            seq_item.overflow = fifo_vif.overflow;

            analysis_port.write(seq_item);
            `uvm_info("RUN_PHASE", seq_item.convert2string(), UVM_HIGH)
        end
    endtask
endclass : fifo_monitor

endpackage : fifo_monitor_pkg
```


Sequencer package:

```
package fifo_sequencer_pkg;
import uvm_pkg::*;
import fifo_seq_item_pkg::*;

`include "uvm_macros.svh"

class fifo_sequencer extends uvm_sequencer #(fifo_seq_item);
    `uvm_component_utils(fifo_sequencer)

    function new(string name = "fifo_sequencer", uvm_component parent = null);
        super.new(name, parent);
    endfunction
endclass : fifo_sequencer

endpackage : fifo_sequencer_pkg
```

Coverage collector package:

```
package fifo_coverage_pkg;
import uvm_pkg::*;
import fifo_seq_item_pkg::*;
import fifo_shared_pkg::*;
`include "uvm_macros.svh"

class fifo_coverage extends uvm_component;
    `uvm_component_utils(fifo_coverage)

    uvm_analysis_export #(fifo_seq_item) analysis_export;
    uvm_tlm_analysis_fifo #(fifo_seq_item) analysis_fifo;
    fifo_seq_item seq_item;

    covergroup cvr_grp;
        option.per_instance = 1;

        wr_en_cp: coverpoint seq_item.wr_en{
            bins wr_en_0 = {0};
            bins wr_en_1 = {1};
            option.weight = 0;
        }

        rd_en_cp: coverpoint seq_item.rd_en{
            bins rd_en_0 = {0};
            bins rd_en_1 = {1};
            option.weight = 0;
        }

        wr_ack_cp: coverpoint seq_item.wr_ack{
            bins wr_ack_0 = {0};
            bins wr_ack_1 = {1};
            option.weight = 0;
        }

        overflow_cp: coverpoint seq_item.overflow{
            bins overflow_0 = {0};
            bins overflow_1 = {1};
            option.weight = 0;
        }

        full_cp: coverpoint seq_item.full{
            bins full_0 = {0};
            bins full_1 = {1};
            option.weight = 0;
        }

        empty_cp: coverpoint seq_item.empty{
```

```

    bins empty_0 = {0};
    bins empty_1 = {1};
    option.weight = 0;
}

almostfull_cp: coverpoint seq_item.almostfull{
    bins almostfull_0 = {0};
    bins almostfull_1 = {1};
    option.weight = 0;
}

almostempty_cp: coverpoint seq_item.almostempty{
    bins almostempty_0 = {0};
    bins almostempty_1 = {1};
    option.weight = 0;
}

underflow_cp: coverpoint seq_item.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(string name = "fifo_coverage", uvm_component parent = null);
    super.new(name, parent);
    cvr_grp = new();
endfunction

function void build_phase(uvm_phase phase);
    super.build_phase(phase);

    analysis_export = new("analysis_export", this);
    analysis_fifo = new("analysis_fifo", this);
endfunction

function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);

    analysis_export.connect(analysis_fifo.analysis_export);
endfunction

task run_phase(uvm_phase phase);
    super.run_phase(phase);
    forever begin
        analysis_fifo.get(seq_item);
        cvr_grp.sample();
    end
endtask

endclass : fifo_coverage

endpackage : fifo_coverage_pkg

```

Scoreboard package:

```
package fifo_scoreboard_pkg;
import uvm_pkg::*;
import fifo_seq_item_pkg::*;
import fifo_shared_pkg::*;
`include "uvm_macros.svh"

class fifo_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(fifo_scoreboard)

    uvm_analysis_export #(fifo_seq_item) analysis_export;
    uvm_tlm_analysis_fifo #(fifo_seq_item) analysis_fifo;
    fifo_seq_item seq_item;

    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 new(string name = "fifo_scoreboard", uvm_component parent =
null);
        super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        analysis_export = new("analysis_export", this);
        analysis_fifo = new("analysis_fifo", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);

        analysis_export.connect(analysis_fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            seq_item = fifo_seq_item::type_id::create("seq_item");
            analysis_fifo.get(seq_item);
```

```

        ref_model(seq_item);
        check_result(seq_item);
    end
endtask

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

        `uvm_error("RUN_PHASE", $sformatf("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 ", seq_item.convert2string(),
data_out_ref, wr_ack_ref, full_ref, empty_ref, almostfull_ref,
almostempty_ref, overflow_ref, underflow_ref));
        error_count++;
    end else begin
        `uvm_info("RUN_PHASE", $sformatf("Comparsion succeeded,
Transaction received from DUT: %s ", seq_item.convert2string()), UVM_HIGH);
        correct_count++;
    end
endtask

function void ref_model(fifo_seq_item seq_item);
    if (!seq_item.rst_n) begin
        fifo_ref.delete();

        wr_ack_ref = 0;
        overflow_ref = 0;
        underflow_ref = 0;
    end else begin
        if (seq_item.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 (seq_item.wr_en) begin

```

```

        if (!full_ref) begin
            fifo_ref.push_back(seq_item.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

function void report_phase(uvm_phase phase);
    super.report_phase(phase);
    `uvm_info("REPORT_PHASE", $sformatf("Total successful transactions:
%d ", correct_count), UVM_MEDIUM);
    `uvm_info("REPORT_PHASE", $sformatf("Total failed transactions: %d ",
error_count), UVM_MEDIUM);
endfunction

endclass : fifo_scoreboard

endpackage : fifo_scoreboard_pkg

```

Configuration package:

```
package fifo_config_obj_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"

class fifo_config_obj extends uvm_object;
    `uvm_object_utils(fifo_config_obj)

    virtual fifo_if fifo_vif;

    function new(string name = "fifo_config_obj");
        super.new(name);
    endfunction
endclass : fifo_config_obj

endpackage : fifo_config_obj_pkg
```


Sequence item package:

```
package fifo_seq_item_pkg;
import uvm_pkg::*;
import fifo_shared_pkg::*;
`include "uvm_macros.svh"

class fifo_seq_item extends uvm_sequence_item;
    `uvm_object_utils(fifo_seq_item)

    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(string name = "fifo_seq_item");
        super.new(name);
    endfunction

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

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

    function string convert2string_stimulus();
        return $sformatf("rst_n = %0b, wr_en = %0b, rd_en = %0b, data_in = 0x%0h",
            rst_n, wr_en, rd_en, data_in);
    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)};
}
endclass : fifo_seq_item

endpackage : fifo_seq_item_pkg
```

Sequence package:

```
package fifo_reset_seq_pkg;
import uvm_pkg::*;
import fifo_seq_item_pkg::*;
import fifo_shared_pkg::*;
`include "uvm_macros.svh"

class fifo_reset_seq extends uvm_sequence #(fifo_seq_item);
    `uvm_object_utils(fifo_reset_seq)

    fifo_seq_item seq_item;

    function new(string name = "fifo_reset_seq");
        super.new(name);
    endfunction

    task body();
        seq_item = fifo_seq_item::type_id::create("seq_item");
        start_item(seq_item);
        seq_item.rst_n = 0;
        seq_item.wr_en = 0;
        seq_item.rd_en = 0;
        seq_item.data_in = {FIFO_WIDTH{1'b0}};
        finish_item(seq_item);
    endtask
endclass : fifo_reset_seq

endpackage : fifo_reset_seq_pkg
```

```

package fifo_write_seq_pkg;
import uvm_pkg::*;
import fifo_seq_item_pkg::*;
import fifo_shared_pkg::*;
`include "uvm_macros.svh"

class fifo_write_seq extends uvm_sequence #(fifo_seq_item);
    `uvm_object_utils(fifo_write_seq)

    fifo_seq_item seq_item;

    function new(string name = "fifo_write_seq");
        super.new(name);
    endfunction

    task body();
        repeat (NUM_TEST_CASES) begin
            seq_item = fifo_seq_item::type_id::create("seq_item");
            seq_item.set_dist(.rd_dist(10), .wr_dist(90));
            start_item(seq_item);
            assert (seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
endclass : fifo_write_seq

endpackage : fifo_write_seq_pkg

```

```

package fifo_read_seq_pkg;
import uvm_pkg::*;
import fifo_seq_item_pkg::*;
import fifo_shared_pkg::*;
`include "uvm_macros.svh"

class fifo_read_seq extends uvm_sequence #(fifo_seq_item);
    `uvm_object_utils(fifo_read_seq)

    fifo_seq_item seq_item;

    function new(string name = "fifo_read_seq");
        super.new(name);
    endfunction

    task body();
        repeat (NUM_TEST_CASES) begin
            seq_item = fifo_seq_item::type_id::create("seq_item");
            seq_item.set_dist(.rd_dist(90), .wr_dist(10));
            start_item(seq_item);
            assert (seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
endclass : fifo_read_seq

endpackage : fifo_read_seq_pkg

```

```

package fifo_read_write_seq_pkg;
import uvm_pkg::*;
import fifo_seq_item_pkg::*;
import fifo_shared_pkg::*;
`include "uvm_macros.svh"

class fifo_read_write_seq extends uvm_sequence #(fifo_seq_item);
    `uvm_object_utils(fifo_read_write_seq)

    fifo_seq_item seq_item;

    function new(string name = "fifo_read_write_seq");
        super.new(name);
    endfunction

    task body();
        repeat (NUM_TEST_CASES) begin
            seq_item = fifo_seq_item::type_id::create("seq_item");
            seq_item.set_dist(.rd_dist(50), .wr_dist(50));
            start_item(seq_item);
            assert (seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
endclass : fifo_read_write_seq

endpackage : fifo_read_write_seq_pkg

```

Shared package:

```
package fifo_shared_pkg;

parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
localparam max_fifo_addr = $clog2(FIFO_DEPTH);

int NUM_TEST_CASES = 500;
int error_count;
int correct_count;

endpackage : fifo_shared_pkg
```

Assertions file:

```
import fifo_shared_pkg::*;

module fifo_sva (fifo_if.DUT fifoif);

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

always_comb begin
    if (!fifoif.rst_n) begin
        assert final(DUT.wr_ptr == 0 && DUT.rd_ptr == 0 && DUT.count == 0);
    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)
        (DUT.count == 0) |> fifoif.empty;
endproperty

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

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



```
endproperty
```

```
property almostempty_p;
```

```
    @(posedge fifoif.clk) disable iff (!fifoif.rst_n)
```

```
        (DUT.count == 1) |-> fifoif.almostempty;
```

```
endproperty
```

```
property wr_ptr_wraparound_p;
```

```
    @(posedge fifoif.clk) disable iff (!fifoif.rst_n)
```

```
        (DUT.wr_ptr == (FIFO_DEPTH - 1) && fifoif.wr_en && !fifoif.full) |=>
```

```
        (DUT.wr_ptr == 0);
```

```
endproperty
```

```
property rd_ptr_wraparound_p;
```

```
    @(posedge fifoif.clk) disable iff (!fifoif.rst_n)
```

```
        (DUT.rd_ptr == FIFO_DEPTH-1 && fifoif.rd_en && !fifoif.empty) |=>
```

```
        (DUT.rd_ptr == 0);
```

```
endproperty
```

```
property ptr_threshold_p;
```

```
    @(posedge fifoif.clk) disable iff (!fifoif.rst_n)
```

```
        ((DUT.wr_ptr < FIFO_DEPTH) && (DUT.rd_ptr < FIFO_DEPTH) && (DUT.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);
```

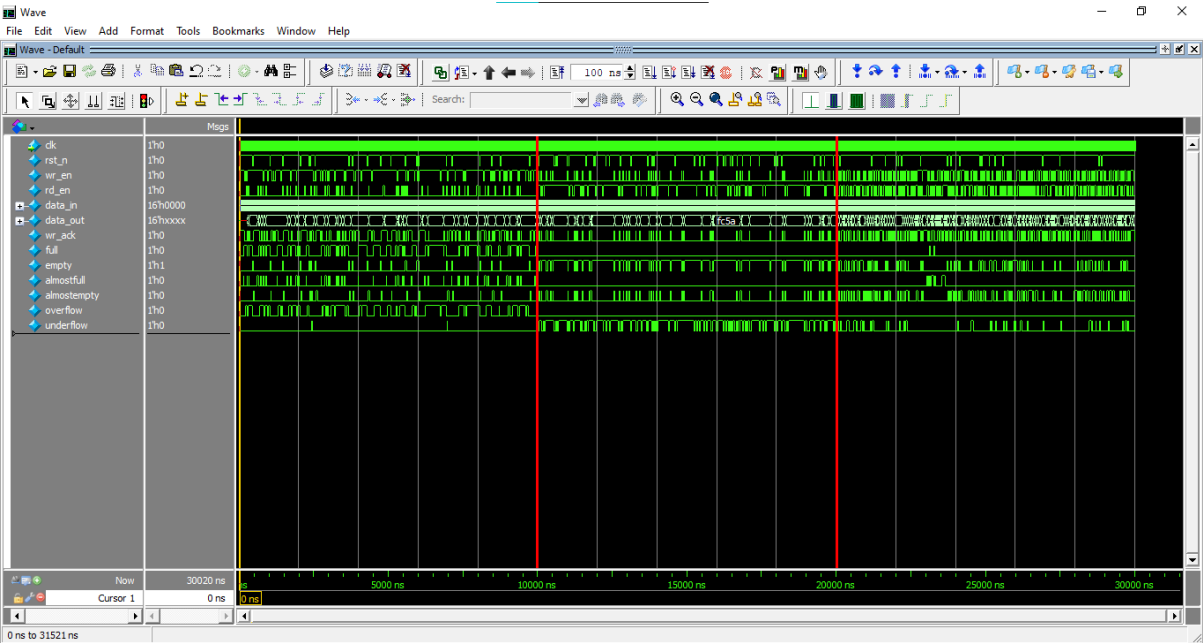
```
endmodule
```

Feature	Assertion
When a write is requested and FIFO is not full , a write acknowledgment must be asserted in the next cycle.	<code>@(posedge clk) disable iff(!rst_n) (wr_en && !full) => wr_ack</code>
When a write is requested while the FIFO is full , an overflow flag must be raised in the next cycle.	<code>@(posedge clk) disable iff(!rst_n) (wr_en && full) => overflow</code>
When a read is requested while the FIFO is empty , an underflow flag must be raised in the next cycle.	<code>@(posedge clk) disable iff(!rst_n) (rd_en && empty) => underflow</code>
The empty flag must be asserted whenever the FIFO count is 0.	<code>@(posedge clk) disable iff(!rst_n) (count == 0) -> empty</code>
The full flag must be asserted whenever the FIFO count equals FIFO_DEPTH.	<code>@(posedge clk) disable iff(!rst_n) (count == FIFO_DEPTH) -> full</code>
The almost full flag must be asserted whenever the FIFO count is one less than full (FIFO_DEPTH - 1).	<code>@(posedge clk) disable iff(!rst_n) (count == FIFO_DEPTH - 1) -> almostfull</code>
The almost empty flag must be asserted whenever the FIFO count equals 1.	<code>@(posedge clk) disable iff(!rst_n) (count == 1) -> almostempty</code>
When the write pointer reaches the last location (FIFO_DEPTH-1) and a valid write occurs, it must wrap around to 0 in the next cycle.	<code>@(posedge clk) disable iff(!rst_n) (wr_ptr == FIFO_DEPTH-1 && wr_en && !full) => (wr_ptr == 0)</code>
When the read pointer reaches the last location (FIFO_DEPTH-1) and a valid read occurs, it must wrap around to 0 in the next cycle.	<code>@(posedge clk) disable iff(!rst_n) (rd_ptr == FIFO_DEPTH-1 && rd_en && !empty) => (rd_ptr == 0)</code>
Both read and write pointers and count must always remain within FIFO bounds (no pointer/count overflow).	<code>@(posedge clk) disable iff(!rst_n) ((wr_ptr < FIFO_DEPTH) && (rd_ptr < FIFO_DEPTH) && (count <= FIFO_DEPTH))</code>

Do file:

```
vlib work  
vlog -f src_files.list +cover -covercells  
vsim -voptargs=+acc work.fifo_top -cover -classdebug -uvmcontrol=all  
add wave /fifo_top/fifoif/*  
coverage save FIFO_tb.ucdb -onexit  
run -all  
quit -sim
```

Simulation snippet:



Transcript:

```
# UVM_INFO FIFO_test.sv(48) @ 0: uvm_test_top [RUN_PHASE] Reset Asserted
# UVM_INFO FIFO_test.sv(49) @ 20: uvm_test_top [RUN_PHASE] Reset Deasserted
# UVM_INFO FIFO_test.sv(52) @ 20: uvm_test_top [RUN_PHASE] Write Only Sequence
Started
# UVM_INFO FIFO_test.sv(54) @ 10020: uvm_test_top [RUN_PHASE] Write Only
Sequence Ended
# UVM_INFO FIFO_test.sv(57) @ 10020: uvm_test_top [RUN_PHASE] Read Only
Sequence Started
# UVM_INFO FIFO_test.sv(59) @ 20020: uvm_test_top [RUN_PHASE] Read Only
Sequence Ended
# UVM_INFO FIFO_test.sv(62) @ 20020: uvm_test_top [RUN_PHASE] Read/Write
Sequence Started
# UVM_INFO FIFO_test.sv(64) @ 30020: uvm_test_top [RUN_PHASE] Read/Write
Sequence Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 30020:
reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO FIFO_scoreboard.sv(106) @ 30020: uvm_test_top.env.scoreboard
[REPORT_PHASE] Total successful transactions: 1501
# UVM_INFO FIFO_scoreboard.sv(107) @ 30020: uvm_test_top.env.scoreboard
[REPORT_PHASE] Total failed transactions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO : 1515
# UVM_WARNING : 0
# UVM_ERROR : 0
# UVM_FATAL : 0
# ** Report counts by id
# [Questa UVM] 2
# [REPORT_PHASE] 2
# [RNTST] 1
# [RUN_PHASE] 1509
# [TEST_DONE] 1
# ** Note: $finish : C:/questasim64_2021.1/win64/./verilog_src/uvm-
1.1d/src/base/uvm_root.svh(430)
# Time: 30020 ns Iteration: 61 Instance: /fifo_top
# Saving coverage database on exit...
# End time: 18:32:47 on Oct 03,2025, Elapsed time: 0:00:39
# Errors: 0, Warnings: 0
```

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:

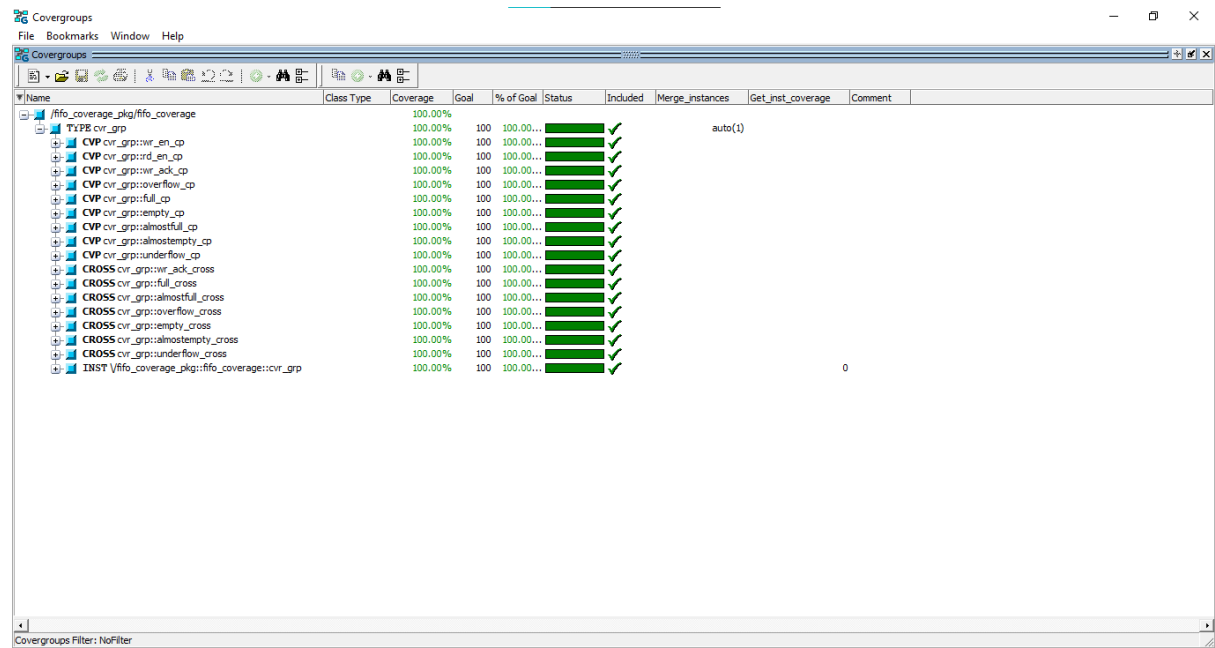
Enabled Coverage	Bins	Hits	Misses	Coverage
-----	----	----	-----	-----
Statements	28	28	0	100.00%

=====

Toggle Coverage:

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

Functional coverage:



Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/rifo_coverage_pkg/rifo_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...					
INST /rifo_coverage_pkg/rifo_coverage::cvr_grp		100.00%	100	100.00...					0

Covergroups Filter: NoFilter

The screenshot displays the 'Assertions' window of a debugger. The window title is 'Assertions'. Below the title bar is a toolbar with various icons for assertion management. The main area contains a table of assertions.

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Include
▲ /vrm_pkg/vrm_req_map::do_write/#ubk#215181159#1731/Immed_17...	Immediate	SVA	on	0	0	-	-	-	-	-	off	assert(\$cast(seq,o)	✗
▲ /vrm_pkg/vrm_req_map::do_read/#ubk#215181159#1771/Immed_17...	Immediate	SVA	on	0	0	-	-	-	-	-	off	assert(\$cast(seq,o)	✗
▲ /fifo_read_write_seq_pkg::fifo_read_seq::body/#ubk#28464967#...	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert(randomize(...))	✓
▲ /fifo_read_seq_pkg::fifo_read_seq::body/#ubk#253944871#17/Immed_...	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert(randomize(...))	✓
▲ /fifo_write_seq_pkg::fifo_write_seq::body/#ubk#26928919#17/Immed_...	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert(randomize(...))	✓
⊕ /fifo_top/DUT/ASSERT/assert_vr_ack	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge ffifo...	✓
⊕ /fifo_top/DUT/ASSERT/assert_overflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge ffifo...	✓
⊕ /fifo_top/DUT/ASSERT/assert_underflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge ffifo...	✓
⊕ /fifo_top/DUT/ASSERT/assert_empty	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge ffifo...	✓
⊕ /fifo_top/DUT/ASSERT/assert_full	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge ffifo...	✓
⊕ /fifo_top/DUT/ASSERT/assert_almostfull	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge ffifo...	✓
⊕ /fifo_top/DUT/ASSERT/assert_almostempty	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge ffifo...	✓
⊕ /fifo_top/DUT/ASSERT/assert_vr_ptr_wraparound	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge ffifo...	✓
⊕ /fifo_top/DUT/ASSERT/assert_rd_ptr_wraparound	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge ffifo...	✓
⊕ /fifo_top/DUT/ASSERT/assert_ptr_threshold	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge ffifo...	✓
▲ /fifo_top/DUT/ASSERT/#ubk#265645057#6/Immed_7	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert(ffifo.empty&&~fif...	✓
▲ /fifo_top/DUT/ASSERT/#ubk#265645057#15/Immed_16	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert(DUT_vr_ptr==0...	✓

At the bottom of the window, there is a status bar that reads 'Assertions Filter: NoFilter'.