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** |
| --- | --- | --- | --- | --- |
| **FIFO1** | When reset is asserted, pointers and flags must return to default values. | Directed at first of simulation then randomized. | - | reset assertion + scoreboard clearing state. |
| **FIFO2** | 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. |
| **FIFO3** | 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. |
| **FIFO4** | 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. |
| **FIFO5** | 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. |
| **FIFO7** | Full = 1 when count = FIFO\_DEPTH. | Fill FIFO until depth reached. | full\_cp. | full assertion, scoreboard checks with full\_ref. |
| **FIFO8** | Empty = 1 when count=0. | Drain FIFO completely. | empty\_cp. | empty assertion, scoreboard checks with empty\_ref. |
| **FIFO9** | Almostfull = 1 when count = FIFO\_DEPTH-1. | Fill until one slot left. | almostfull\_cp. | almostfull assertion, scoreboard checks with almostfull\_ref. |
|  |  |  |  |  |
| **FIFO10** | Almostempty = 1 when count = 1. | Drain until one entry left. | almostempty\_cp. | almostempty assertion, scoreboard checks with almostempty\_ref. |
| **FIFO11** | 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. |
| **FIFO12** | 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. |
| **FIFO13** | Count should never exceed FIFO\_DEPTH or go negative. | Random sequences of writes/reads. | - | ptr\_threshold assertion. |

UVM structure:

صورة تحتوي على نص, لقطة شاشة, عرض, مستطيل

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

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).

1. **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.

1. **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.

1. **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.

1. **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

1. **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.

1. **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);

        // 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 successed, 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: %0d ", correct\_count), UVM\_MEDIUM);

        `uvm\_info("REPORT\_PHASE", $sformatf("Total failed transactions: %0d ", 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. | @(posedge clk) disable iff(!rst\_n) (wr\_en && !full) |=> wr\_ack |
| When a write is requested while the FIFO is **full**, an **overflow flag** must be raised in the next cycle. | @(posedge clk) disable iff(!rst\_n) (wr\_en && full) |=> overflow |
| When a read is requested while the FIFO is **empty**, an **underflow flag** must be raised in the next cycle. | @(posedge clk) disable iff(!rst\_n) (rd\_en && empty) |=> underflow |
| The **empty flag** must be asserted whenever the FIFO count is 0. | @(posedge clk) disable iff(!rst\_n) (count == 0) |-> empty |
| The **full flag** must be asserted whenever the FIFO count equals FIFO\_DEPTH. | @(posedge clk) disable iff(!rst\_n) (count == FIFO\_DEPTH) |-> full |
| The **almost full flag** must be asserted whenever the FIFO count is one less than full (FIFO\_DEPTH - 1). | @(posedge clk) disable iff(!rst\_n) (count == FIFO\_DEPTH - 1) |-> almostfull |
| The **almost empty flag** must be asserted whenever the FIFO count equals 1. | @(posedge clk) disable iff(!rst\_n) (count == 1) |-> almostempty |
| 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. | @(posedge clk) disable iff(!rst\_n) (wr\_ptr == FIFO\_DEPTH-1 && wr\_en && !full) |=> (wr\_ptr == 0) |
| 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. | @(posedge clk) disable iff(!rst\_n) (rd\_ptr == FIFO\_DEPTH-1 && rd\_en && !empty) |=> (rd\_ptr == 0) |
| Both read and write pointers and count must always remain within FIFO bounds (no pointer/count overflow). | @(posedge clk) disable iff(!rst\_n) ((wr\_ptr < FIFO\_DEPTH) && (rd\_ptr < FIFO\_DEPTH) && (count <= FIFO\_DEPTH)) |

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:

صورة تحتوي على نص, لقطة شاشة, عرض, برمجيات

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

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

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:

صورة تحتوي على نص, لقطة شاشة, عرض, برمجيات

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

Assertions coverage:

صورة تحتوي على نص, لقطة شاشة, برمجيات, أيقونة الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.