Faculty of Engineering
Cairo University

Project
UVM

Submitted to: Eng. Kareem Waseem

Eng. Abdelrahman Sherif

Submitted by: Seif Eldin Ahmed Hassan

Table of Contents

1)Overview

The Synchronous FIFO design in this project is a First-In-First-Out (FIFO) memory buffer. It allows data to be written into a queue and read from it in the same sequential order, ensuring smooth communication between processes operating on the same clock domain.

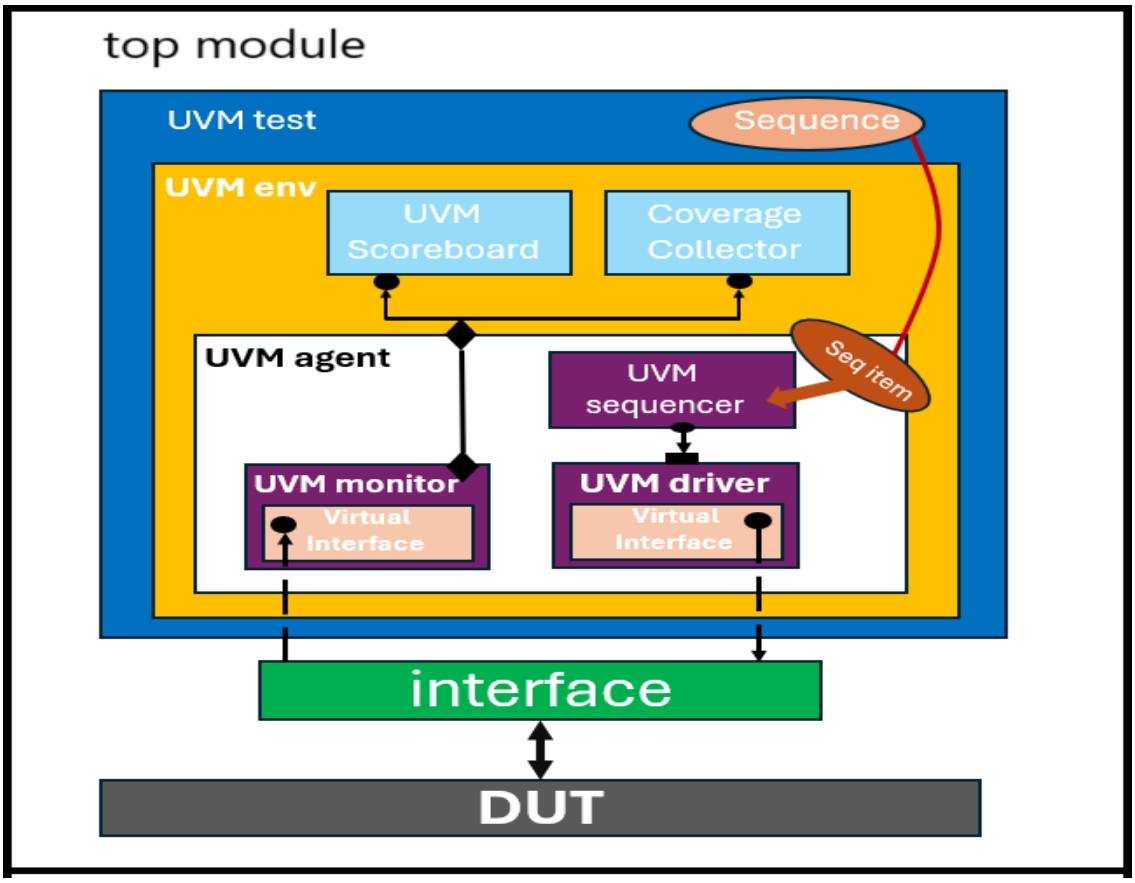
Key Features:

- FIFO_WIDTH: Defines the width of data input and output buses, as well as the memory word width (default: 16 bits).
- FIFO_DEPTH: Determines the depth of the FIFO, representing the number of data entries that can be stored (default: 8 entries).

FIFO Signals:

Signal	Direction	Description
data_in	Input	Write Data: The input data bus used when writing to the FIFO.
wr_en	Input	Write Enable: Enables data writing when the FIFO is not full.
rd_en	Input	Read Enable: Enables data reading when the FIFO is not empty.
clk	Input	Clock: The clock signal for synchronizing operations.
rst_n	Input	Reset: Active-low asynchronous reset signal.
data_out	Output	Read Data: The data output bus when reading from the FIFO.
full	Output	Full Flag: Indicates the FIFO is full and cannot accept more data.
almostfull	Output	Almost Full: Indicates one more write can be performed before full.
empty	Output	Empty Flag: Indicates the FIFO is empty.
almostempty	Output	Almost Empty: Indicates one more read can be performed before empty.
overflow	Output	Overflow: Indicates a rejected write operation due to full FIFO.
underflow	Output	Underflow: Indicates a rejected read operation due to empty FIFO.
wr_ack	Output	Write Acknowledge: Indicates a successful write operation.

2)UVM Structure



3)UVM Flow

1. **Top Module**:
   - The starting point, where the UVM test is invoked.
   - Generates the clock, instantiates the DUT, imports test packages, binds assertions, and sets the virtual interface in the UVM configuration database.
   - Runs the test.
2. **UVM Test**:
   - Builds the environment and sequences.
   - Retrieves the virtual interface and configuration object from the database.
   - Starts sequences on the sequencer.
3. **UVM Sequence**:
   - Core stimulus generator of the verification environment.
   - Creates multiple sequence items.
   - It contains reset_sequence, write_only_sequence, read_only_sequence & write_read_sequence.

4. **UVM Sequence Item**:
    - Contains the data for communication with the DUT.
    - Generates random stimuli based on constraints.
5. **UVM Environment**:
    - Builds and connects the UVM agent, scoreboard, and coverage collector.
6. **UVM Agent**:
    - Builds the monitor, sequencer, and driver.
    - Links the driver and sequencer.
    - Retrieves the configuration object and assigns its virtual interface to the driver and monitor.
7. **UVM Sequencer**:
    - Acts as a FIFO to store and deliver sequence items (transactions) from the sequence to the driver.
8. **UVM Driver**:
    - Pulls sequence items from the sequencer and assigns them to the virtual interface, directly interacting with the DUT.
9. **UVM Monitor**:
    - Collects DUT signals from the interface and converts them into sequence items.
    - Sends these items to analysis components.
10. **UVM Analysis Components**:

    - **Scoreboard**: Compares sequence items with the reference model to verify functionality.
    - **Functional Coverage Collector**: Samples sequence items for functional coverage.

4) Verification Plan

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	When the reset is asserted, the internal pointers, counters, overflow and underflow	Directed at the start of the simulation, then randomized with constraint that drive the reset to be off most of the simulation time	cover the reset signal	A checker in the scoreboard to check the internal signals and counter - Other signals are checked with assertions in the DUT module
FIFO_2	write only sequence	constraing writing only and no read during simulation	Cross coverage between signals write enable, read enable and each output control signals	A checker in the scoreboard to check the internal signals and counter - Other signals are checked with assertions in the DUT module
FIFO_3	read only sequence	constraing reading only and no write during simulation	Cross coverage between signals write enable, read enable and each output control signals	A checker in the scoreboard to check the internal signals and counter - Other signals are checked with assertions in the DUT module
FIFO_4	Verify the functionality of the FIFO with constrained randomization	Randomization in a repeat 9999 iterations with constraints on the reset to be off 90% of the time, write enable signal to be high 70% of the time, read enable signal to be active 30% of the time.	Cross coverage between signals write enable, read enable and each output control signals	A checker in the scoreboard to check the internal signals and counter - Other signals are checked with assertions in the DUT module

5) Bugs

- Comment: All bugs are commented in the design snippet

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

6) Assertions

Feature	Assertions

When the reset is asserted, the flags and pointers will reset to zero	`rst_count_assert: assert final (count == 0);`
	`rst_wr_ptr_assert: assert final (wr_ptr == 0);`
	`rst_rd_ptr_assert: assert final (rd_ptr == 0);`

When the FIFO is full (count equals FIFO depth), the full flag will be asserted	`full_assert: assert final (full);`

When the FIFO is empty (count equals zero), the empty flag will be asserted	`empty_assert: assert final (empty);`
When count equals FIFO depth - 1, the almost full flag will be asserted	`almostfull_assert: assert final (almostfull);`
When count equals 1, the almost empty flag will be asserted	`almostempty_assert: assert final (almostempty);`
When write is enabled and FIFO is not full, acknowledgment will be sent	`wr_ack_assert: assert final (wr_ack);`
When count equals FIFO depth and write is enabled, overflow will occur	`overflow_assert: assert final (overflow);`
When count equals 0 and read is enabled, underflow will occur	`underflow_assert: assert final (underflow);`
When write is enabled and FIFO is not full, the write pointer will increment	`wr_ptr_assert: assert property @(posedge clk)(wr_en && count != FIFO_DEPTH) |=> ($past(wr_ptr) + 1'b1 == wr_ptr);`
When read is enabled and FIFO is not empty, the read pointer will increment	`rd_ptr_assert: assert property @(posedge clk)(rd_en && count != 0) |=> ($past(rd_ptr) + 1'b1 == rd_ptr);`
When write is enabled but no read, the count will increment	`wr_count_assert: assert property @(posedge clk)(!rd_en && wr_en && count != FIFO_DEPTH) |=> ($past(count) + 1'b1 == count);`
When read is enabled but no write, the count will decrement	`rd_count_assert: assert property @(posedge clk) (rd_en && !wr_en && count != 0) |=> ($past(count) - 1'b1 == count);`
When both read and write are enabled, and FIFO is empty, the count will increment	`rd_wr_count_empty_assert: assert property @(posedge clk) (rd_en && wr_en && empty) |=> ($past(count) + 1'b1 == count);`
When both read and write are enabled, and FIFO is full, the count will decrement	`rd_wr_count_full_assert: assert property @(posedge clk) (rd_en && wr_en && full) |=> ($past(count) - 1'b1 == count);`

7) Codes

a) RTL

```
////////////////////////////////////////////////////////////////////////////////
// Author: Kareem Waseem
// Course: Digital Verification using SV & UVM
//
// Description: FIFO Design
//
////////////////////////////////////////////////////////////////////////////////
module FIFO(FIFO_if.DUT FIFOif);
localparam max_fifo_addr = $clog2(FIFOif.FIFO_DEPTH);
reg [FIFOif.FIFO_WIDTH-1:0] mem [FIFOif.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;
    end
    else if (FIFOif.wr_en && count < FIFOif.FIFO_DEPTH) begin
        mem[wr_ptr] <= FIFOif.data_in;
        // FIFOif.wr_ack <= 1; // BUG HERE
        wr_ptr <= wr_ptr + 1;
    end
    // BUG HERE
    // 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;
    end
    else if (FIFOif.rd_en && count != 0) begin
        FIFOif.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    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) ||
        (({FIFOif.wr_en, FIFOif.rd_en} == 2'b11) && FIFOif.empty)) // Added the
part for the note
            count <= count + 1;
        else if ((({FIFOif.wr_en, FIFOif.rd_en} == 2'b01) && !FIFOif.empty) ||
        (({FIFOif.wr_en, FIFOif.rd_en} == 2'b11) && FIFOif.full)) // Added the part
for the note
            count <= count - 1;
    end
end
assign FIFOif.full = (count == FIFOif.FIFO_DEPTH)? 1 : 0;
assign FIFOif.empty = (count == 0)? 1 : 0;
assign FIFOif.underflow = (FIFOif.empty && FIFOif.rd_en)? 1 : 0;
// assign FIFOif.almostfull = (count == FIFOif.FIFO_DEPTH-2)? 1 : 0; // BUG HERE
assign FIFOif.almostempty = (count == 1)? 1 : 0;
// Fixed bugs
assign FIFOif.wr_ack = ((count != FIFOif.FIFO_DEPTH) && FIFOif.wr_en)? 1 : 0;
assign FIFOif.almostfull = (count == FIFOif.FIFO_DEPTH - 1)? 1 : 0;
assign FIFOif.overflow = (FIFOif.full && FIFOif.wr_en)? 1 : 0;
endmodule
```

b) SVA

```
module FIFO_SVA (
    FIFO_if.DUT FIFOif
);
    property write_count;
        @(posedge FIFOif.clk) disable iff(!FIFOif.rst_n)
        (!FIFOif.rd_en && FIFOif.wr_en && dut.count != FIFOif.FIFO_DEPTH) |=>
($past(dut.count) + 1'b1 == dut.count);
    endproperty
    property read_count;
        @(posedge FIFOif.clk) disable iff(!FIFOif.rst_n)
        (FIFOif.rd_en && !FIFOif.wr_en && dut.count != 0) |=> ($past(dut.count) -
1'b1 == dut.count);
    endproperty
    property read_write_count;
        @(posedge FIFOif.clk) disable iff(!FIFOif.rst_n)
        (FIFOif.rd_en && FIFOif.wr_en && dut.count != 0 && dut.count !=
FIFOif.FIFO_DEPTH) |=> ($past(dut.count) == dut.count);
    endproperty
    property read_write_count_empty;
        @(posedge FIFOif.clk) disable iff(!FIFOif.rst_n)
        (FIFOif.rd_en && FIFOif.wr_en && FIFOif.empty) |=> ($past(dut.count) + 1'b1
== dut.count);
```

```
    endproperty
    property read_write_count_full;
        @(posedge FIFOif.clk) disable iff(!FIFOif.rst_n)
        (FIFOif.rd_en && FIFOif.wr_en && FIFOif.full) |=> ($past(dut.count) - 1'b1
== dut.count);
    endproperty
    property write_ptr;
        @(posedge FIFOif.clk) disable iff (!FIFOif.rst_n)
        (FIFOif.wr_en && dut.count != FIFOif.FIFO_DEPTH) |=> ($past(dut.wr_ptr) +
1'b1 == dut.wr_ptr);
    endproperty
    property read_ptr;
        @(posedge FIFOif.clk) disable iff (!FIFOif.rst_n)
        (FIFOif.rd_en && dut.count != 0) |=> ($past(dut.rd_ptr) + 1'b1 ==
dut.rd_ptr);
    endproperty
    // Assertions
    wr_count_assert: assert property (write_count)
        else $error("Assertion failed: dut.count did not increment when write
correctly at time %0t, dut.count = %0d, expected = %0d",
                        $time, dut.count, $past(dut.count) + 1'b1);
    rd_count_assert: assert property (read_count)
        else $error("Assertion failed: dut.count did not decrement when read
correctly at time %0t, dut.count = %0d, expected = %0d",
                        $time, dut.count, $past(dut.count) - 1'b1);
    rd_wr_count_assert: assert property (read_write_count)
        else $error("Assertion failed: dut.count should remain the same when read
and write, and not empty or full at time %0t, dut.count = %0d, expected = %0d",
                        $time, dut.count, $past(dut.count));
    rd_wr_count_empty_assert:assert property (read_write_count_empty)
        else $error("Assertion failed: dut.count did not increment when write, read
and empty correctly at time %0t, dut.count = %0d, expected = %0d",
                        $time, dut.count, $past(dut.count) + 1'b1);
    rd_wr_count_full_assert:assert property (read_write_count_full)
        else $error("Assertion failed: dut.count did not increment when write, read
and full correctly at time %0t, dut.count = %0d, expected = %0d",
                        $time, dut.count, $past(dut.count) - 1'b1);
    wr_ptr_assert: assert property (write_ptr)
        else $error("Assertion failed: dut.wr_ptr did not increment correctly at
time %0t, dut.wr_ptr = %0d, expected = %0d",
                        $time, dut.wr_ptr, $past(dut.wr_ptr) + 1'b1);
    rd_ptr_assert: assert property (read_ptr)
        else $error("Assertion failed: dut.rd_ptr did not increment correctly at
time %0t, dut.rd_ptr = %0d, expected = %0d",
                        $time, dut.rd_ptr, $past(dut.rd_ptr) + 1'b1);
    wr_count_cover: cover property (write_count);
```

```
    rd_count_cover: cover property (read_count);
    rd_wr_count_cover: cover property (read_write_count);
    rd_wr_count_empty_cover:cover property (read_write_count_empty);
    rd_wr_count_full_cover:cover property (read_write_count_full);
    wr_ptr_cover: cover property (write_ptr);
    rd_ptr_cover: cover property (read_ptr);
    always_comb begin
        if(!FIFOif.rst_n)begin
            rst_count_assert: assert final (dut.count==0);
            rst_count_cover: cover final (dut.count==0);
            rst_wr_ptr_assert: assert final (dut.wr_ptr==0);
            rst_wr_ptr_cover: cover final (dut.wr_ptr==0);
            rst_rd_ptr_assert: assert final (dut.rd_ptr==0);
            rst_rd_ptr_cover: cover final (dut.rd_ptr==0);
        end
        if(dut.count == FIFOif.FIFO_DEPTH) begin
            full_assert: assert final (FIFOif.full);
            full_cover: cover final (FIFOif.full);
        end
        if(dut.count == 0) begin
            empty_assert: assert final (FIFOif.empty);
            empty_cover: cover final (FIFOif.empty);
        end
        if(dut.count == 0 && FIFOif.rd_en) begin
            underflow_assert: assert final (FIFOif.underflow);
            underflow_cover: cover final (FIFOif.underflow);
        end
        if(dut.count == FIFOif.FIFO_DEPTH && FIFOif.wr_en) begin
            overflow_assert: assert final (FIFOif.overflow);
            overflow_cover: cover final (FIFOif.overflow);
        end
        if(dut.count == FIFOif.FIFO_DEPTH - 1) begin
            almostfull_assert: assert final (FIFOif.almostfull);
            almostfull_cover: cover final (FIFOif.almostfull);
        end
        if(dut.count == 1) begin
            almostempty_assert: assert final (FIFOif.almostempty);
            almostempty_cover: cover final (FIFOif.almostempty);
        end
        if((dut.count != FIFOif.FIFO_DEPTH) && FIFOif.wr_en) begin
            wr_ack_assert: assert final (FIFOif.wr_ack);
            wr_ack_cover: cover final (FIFOif.wr_ack);
        end
    end
endmodule
```

c) Interface

```
interface FIFO_if (input bit clk);
  parameter FIFO_WIDTH = 16;
  parameter FIFO_DEPTH = 8;
  // Input Signals
  logic [FIFO_WIDTH-1:0] data_in;
  logic rst_n, wr_en, rd_en;
  // Output signals
  logic [FIFO_WIDTH-1:0] data_out;
  logic wr_ack, overflow;
  logic full, empty, almostfull, almostempty, underflow;
  modport DUT(
      input clk, data_in, rst_n, wr_en, rd_en,
      output data_out, wr_ack, overflow, full, empty, almostfull, almostempty,
underflow
  );
endinterface
```

d) Agent

```
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 sqr;
        FIFO_driver drv;
        FIFO_monitor mon;
        FIFO_config_obj FIFO_cfg_agent;
        uvm_analysis_port #(FIFO_seq_item) agt_ap;
        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_agent))
                `uvm_fatal("build_phase" , "Unable to get configuration object")
            //create the driver, sequencer and monitor
            sqr = FIFO_sequencer::type_id::create("sqr", this);
```

```
            drv = FIFO_driver::type_id::create("drv", this);
            mon = FIFO_monitor::type_id::create("mon", this);
            agt_ap = new("agt_ap", this);
        endfunction
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            //connect the vif of the driver and monitor
            drv.FIFO_driver_vif = FIFO_cfg_agent.FIFO_config_vif;
            mon.FIFO_monitor_vif = FIFO_cfg_agent.FIFO_config_vif;
            //connect the ports for the sqr and driver
            drv.seq_item_port.connect(sqr.seq_item_export);
            //connect the analysis port of the monitor and agent
            mon.mon_ap.connect(agt_ap);
        endfunction
    endclass
endpackage
```

e) Configuration Object

```
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_config_vif;
        function new(string name = "FIFO_config_obj");
            super.new(name);
        endfunction
    endclass
endpackage
```

f) Coverage Collector

```
package FIFO_coverage_pkg;
    import uvm_pkg::*;
    import FIFO_seq_item_pkg::*;
    `include "uvm_macros.svh"
    class FIFO_coverage extends uvm_component;
        `uvm_component_utils(FIFO_coverage)
        uvm_analysis_export #(FIFO_seq_item) cov_export;
        uvm_tlm_analysis_fifo #(FIFO_seq_item) cov_fifo;
        FIFO_seq_item seq_item_cov;
        covergroup cg_FIFO;
            cp_wr_en: coverpoint seq_item_cov.wr_en;
            cp_rd_en: coverpoint seq_item_cov.rd_en;
```

```
        cp_wr_ack: coverpoint seq_item_cov.wr_ack;
        cp_overflow: coverpoint seq_item_cov.overflow;
        cp_full: coverpoint seq_item_cov.full;
        cp_empty: coverpoint seq_item_cov.empty;
        cp_almostfull: coverpoint seq_item_cov.almostfull;
        cp_almostempty: coverpoint seq_item_cov.almostempty;
        cp_underflow: coverpoint seq_item_cov.underflow;
        // Cross coverage
        // Write Ack cross with ignored bins
        cx_wr_ack: cross cp_wr_en, cp_rd_en, cp_wr_ack {
            ignore_bins auto_wr_ack_0 = binsof(cp_wr_en) intersect {0} &&
                                        binsof(cp_rd_en) intersect {1} &&
                                        binsof(cp_wr_ack) intersect {1};
            ignore_bins auto_wr_ack_1 = binsof(cp_wr_en) intersect {0} &&
                                        binsof(cp_rd_en) intersect {0} &&
                                        binsof(cp_wr_ack) intersect {1};
            ignore_bins auto_wr_ack_2 = binsof(cp_wr_en) intersect {1} &&
                                        binsof(cp_rd_en) intersect {1} &&
                                        binsof(cp_wr_ack) intersect {0};
        }
        // Overflow cross with ignored bins
        cx_overflow: cross cp_wr_en, cp_rd_en, cp_overflow {
            ignore_bins auto_overflow_0 = binsof(cp_wr_en) intersect {0} &&
                                        binsof(cp_rd_en) intersect {1} &&
                                        binsof(cp_overflow) intersect {1};
            ignore_bins auto_overflow_1 = binsof(cp_wr_en) intersect {0} &&
                                        binsof(cp_rd_en) intersect {0} &&
                                        binsof(cp_overflow) intersect {1};
            ignore_bins auto_overflow_2 = binsof(cp_wr_en) intersect {1} &&
                                        binsof(cp_rd_en) intersect {1} &&
                                        binsof(cp_overflow) intersect {1};
        }
        // Full cross with ignored bins
        cx_full: cross cp_wr_en, cp_rd_en, cp_full {
            ignore_bins auto_full_0 = binsof(cp_wr_en) intersect {0} &&
                                      binsof(cp_rd_en) intersect {1} &&
                                      binsof(cp_full) intersect {1};
            ignore_bins auto_full_1 = binsof(cp_wr_en) intersect {1} &&
                                      binsof(cp_rd_en) intersect {1} &&
                                      binsof(cp_full) intersect {1};
        }
        // Empty cross coverage
        cx_empty: cross cp_wr_en, cp_rd_en, cp_empty;
        // Almost full cross coverage
        cx_almostfull: cross cp_wr_en, cp_rd_en, cp_almostfull;
        // Almost empty cross coverage
```

```
            cx_almostempty: cross cp_wr_en, cp_rd_en, cp_almostempty;
            // Underflow cross with ignored bins
            cx_underflow: cross cp_wr_en, cp_rd_en, cp_underflow {
                ignore_bins auto_underflow_0 = binsof(cp_wr_en) intersect {1} &&
                                               binsof(cp_rd_en) intersect {0} &&
                                               binsof(cp_underflow) intersect {1};
                ignore_bins auto_underflow_1 = binsof(cp_wr_en) intersect {0} &&
                                               binsof(cp_rd_en) intersect {0} &&
                                               binsof(cp_underflow) intersect {1};
            }
        endgroup: cg_FIFO
        function new(string name = "FIFO_coverage", uvm_component parent = null);
            super.new(name, parent);
            cg_FIFO = new();
            cg_FIFO.start();
        endfunction
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            cov_export = new("cov_export", this);
            cov_fifo = new("cov_fifo", this);
        endfunction
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            //connect the analysis exports
            cov_export.connect(cov_fifo.analysis_export);
        endfunction
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo.get(seq_item_cov);
                cg_FIFO.sample();
            end
        endtask
    endclass
endpackage
```

g) Driver

```
package FIFO_driver_pkg;
    import uvm_pkg::*;
    import FIFO_config_obj_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_driver_vif;
```

```
        FIFO_config_obj FIFO_cfg_driver;
        FIFO_seq_item stim_seq_item;
        function new(string name = "FIFO_driver", uvm_component parent = null);
            super.new(name, parent);
        endfunction
        // Build phase to retrieve virtual interface
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            // Retrieve the virtual interface from uvm_config_db
            if(!uvm_config_db #(FIFO_config_obj)::get(this, "", "CFG",
FIFO_cfg_driver)) begin
                `uvm_fatal("build_phase", "Driver - Unable to get the virtual
interface of the FIFO from uvm_config_db");
            end
        endfunction
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                stim_seq_item = FIFO_seq_item::type_id::create("stim_seq_item");
                seq_item_port.get_next_item(stim_seq_item);
                FIFO_driver_vif.rst_n = stim_seq_item.rst_n;
                FIFO_driver_vif.data_in = stim_seq_item.data_in;
                FIFO_driver_vif.wr_en = stim_seq_item.wr_en;
                FIFO_driver_vif.rd_en = stim_seq_item.rd_en;
                @(negedge FIFO_driver_vif.clk);
                seq_item_port.item_done();
                `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(),
UVM_HIGH)
            end
        endtask
    endclass
endpackage
```

h) Environment

```
package FIFO_env_pkg;
    import uvm_pkg::*;
    import FIFO_scoreboard_pkg::*;
    import FIFO_coverage_pkg::*;
    import FIFO_agent_pkg::*;
    `include "uvm_macros.svh"
    class FIFO_env extends uvm_env;
        `uvm_component_utils(FIFO_env)
        FIFO_agent agt;
        FIFO_scoreboard sb;
        FIFO_coverage cov;
```

```
        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);
            agt = FIFO_agent::type_id::create("agt", this);
            sb = FIFO_scoreboard::type_id::create("sb", this);
            cov = FIFO_coverage::type_id::create("cov", this);
        endfunction
        function void connect_phase(uvm_phase phase);
            agt.agt_ap.connect(sb.sb_export);
            agt.agt_ap.connect(cov.cov_export);
        endfunction
    endclass
endpackage
```

i) Monitor

```
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_monitor_vif;
        FIFO_seq_item stim_seq_item;
        uvm_analysis_port #(FIFO_seq_item) mon_ap;
        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);
            mon_ap = new("mon_ap", this);
        endfunction
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                stim_seq_item = FIFO_seq_item::type_id::create("stim_seq_item");
                @(negedge FIFO_monitor_vif.clk);
                stim_seq_item.rst_n = FIFO_monitor_vif.rst_n;
                stim_seq_item.data_in = FIFO_monitor_vif.data_in;
                stim_seq_item.wr_en = FIFO_monitor_vif.wr_en;
                stim_seq_item.rd_en = FIFO_monitor_vif.rd_en;

                stim_seq_item.data_out = FIFO_monitor_vif.data_out;
                stim_seq_item.wr_ack = FIFO_monitor_vif.wr_ack;
```

```
                stim_seq_item.overflow = FIFO_monitor_vif.overflow;
                stim_seq_item.full = FIFO_monitor_vif.full;
                stim_seq_item.empty = FIFO_monitor_vif.empty;
                stim_seq_item.almostfull = FIFO_monitor_vif.almostfull;
                stim_seq_item.almostempty = FIFO_monitor_vif.almostempty;
                stim_seq_item.underflow = FIFO_monitor_vif.underflow;
                mon_ap.write(stim_seq_item);
                `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(),
UVM_HIGH)
            end
        endtask
    endclass
endpackage
```

j) Scoreboard

```
package FIFO_scoreboard_pkg;
    import uvm_pkg::*;
    import FIFO_seq_item_pkg::*;
    `include "uvm_macros.svh"
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;
    class FIFO_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(FIFO_scoreboard)
        uvm_analysis_export #(FIFO_seq_item) sb_export;
        uvm_tlm_analysis_fifo #(FIFO_seq_item) sb_fifo;
        FIFO_seq_item seq_item_sb;
        localparam max_fifo_addr = $clog2(FIFO_DEPTH);
        bit [FIFO_WIDTH-1:0] data_out_ref;
        bit [FIFO_WIDTH - 1 : 0] fifo_mem[FIFO_DEPTH-1:0];
        bit [max_fifo_addr - 1 : 0] write_ptr, read_ptr;
        bit [max_fifo_addr : 0] count;
        int error_count = 0;
        int correct_count = 0;
        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);
            sb_export = new("sb_export", this);
            sb_fifo = new("sb_fifo", this);
        endfunction
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            sb_export.connect(sb_fifo.analysis_export);
```

```
        endfunction
        // Task to calculate the reference model output based on the seq_item_sb
        task ref_model();
            if (!seq_item_sb.rst_n) begin
                read_ptr = 0;
                write_ptr = 0;
                count = 0;
            end
            else begin
                if (seq_item_sb.rd_en && count != 0) begin
                    data_out_ref = fifo_mem[read_ptr];
                    read_ptr++;
                end
                if (seq_item_sb.wr_en && count != FIFO_DEPTH) begin
                    fifo_mem[write_ptr] = seq_item_sb.data_in;
                    write_ptr++;
                end
                if((seq_item_sb.wr_en && !seq_item_sb.rd_en && count != FIFO_DEPTH)
||
                (seq_item_sb.wr_en && seq_item_sb.rd_en && count == 0))begin
                    count++;
                end
                if((!seq_item_sb.wr_en && seq_item_sb.rd_en && count != 0) ||
                (seq_item_sb.wr_en && seq_item_sb.rd_en && count == FIFO_DEPTH))
begin
                    count--;
                end
            end
        endtask
        // Task to check the data_out from DUT against the reference model
(data_out_ref)
        task check_data();
            ref_model();  // Call ref_model to compute expected data_out_ref
            // Now compare the reference model data_out_ref with DUT data_out
            if (seq_item_sb.data_out !== data_out_ref) begin
                `uvm_error("check_data", $sformatf("Error: data_out mismatch!
Expected: %0h, Got: %0h", data_out_ref, seq_item_sb.data_out))
                error_count++;
            end else begin
                `uvm_info("check_data", $sformatf("Correct: data_out matches at
time %0t!", $time), UVM_HIGH)
                correct_count++;
            end
        endtask
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
```

```
            forever begin
                sb_fifo.get(seq_item_sb);
                // Call the check_data task to check each transaction
                check_data();
            end
        endtask
        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
endpackage
```

k) Sequence Item

```
package FIFO_seq_item_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    parameter FIFO_WIDTH = 16;
    class FIFO_seq_item extends uvm_sequence_item;
        `uvm_object_utils(FIFO_seq_item)
        // Group: Variables
        rand bit [FIFO_WIDTH-1:0] data_in;
        rand bit rst_n, wr_en, rd_en;
        bit [FIFO_WIDTH-1:0] data_out;
        bit wr_ack, overflow;
        bit full, empty, almostfull, almostempty, underflow;
        int RD_EN_ON_DIST;
        int WR_EN_ON_DIST;
        // Constructor: new
        function new(string name = "FIFO_seq_item", int RD_EN_ON_DIST = 30, int
WR_EN_ON_DIST = 70);
            super.new(name);
            this.RD_EN_ON_DIST = RD_EN_ON_DIST;
            this.WR_EN_ON_DIST = WR_EN_ON_DIST;
        endfunction: new
        // Group: Functions
        function string convert2string();
          string s;
          s = super.convert2string();
          return $sformatf("%s data_in = 0b%0b, rst_n = 0b%0b, wr_en = 0b%0b, rd_en
= 0b%0b, data_out = 0b%0b, wr_ack = 0b%0b, overflow = 0b%0b, full = 0b%0b, empty =
0b%0b, almostfull = 0b%0b, almostempty = 0b%0b, underflow = 0b%0b",
```

```
              s, data_in, rst_n, wr_en, rd_en, data_out, wr_ack, overflow, full,
empty, almostfull, almostempty, underflow);
        endfunction: convert2string
        function string convert2string_stimulus();
          return $sformatf("data_in = 0b%0b, rst_n = 0b%0b, wr_en = 0b%0b, rd_en =
0b%0b, data_out = 0b%0b, wr_ack = 0b%0b, overflow = 0b%0b, full = 0b%0b, empty =
0b%0b, almostfull = 0b%0b, almostempty = 0b%0b, underflow = 0b%0b",
              data_in, rst_n, wr_en, rd_en, data_out, wr_ack, overflow, full,
empty, almostfull, almostempty, underflow);
        endfunction: convert2string_stimulus
        // Group: Constraints
        // General reset constraint
        constraint reset_con {
            rst_n dist {
              0 :/ 10,
              1 :/ 90
            };
          }
          constraint wr_en_con {
            wr_en dist {
              1 :/ WR_EN_ON_DIST,
              0 :/ (100 - WR_EN_ON_DIST)
            };
          }
          constraint rd_en_con {
            rd_en dist {
              1 :/ RD_EN_ON_DIST,
              0 :/ (100 - RD_EN_ON_DIST)
            };
          }
    endclass: FIFO_seq_item
endpackage
```

l) Sequence

```
package FIFO_sequence_pkg;
    import uvm_pkg::*;
    import FIFO_seq_item_pkg::*;
    `include "uvm_macros.svh"
    class FIFO_reset_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_reset_sequence)
        FIFO_seq_item seq_item;
        function new(string name = "FIFO_reset_sequence");
            super.new(name);
        endfunction: new
        task body();
```

```
            seq_item = FIFO_seq_item::type_id::create("seq_item");
            start_item(seq_item);
            seq_item.rst_n = 0;
            seq_item.data_in = 0;
            seq_item.wr_en = 0;
            seq_item.rd_en = 0;
            finish_item(seq_item);
        endtask
    endclass: FIFO_reset_sequence
    class FIFO_write_only_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_write_only_sequence)
        FIFO_seq_item seq_item;
        int read_dist = 0 ; // No read
        int write_dist = 100 ; // Write only
        function new(string name = "FIFO_write_only_sequence");
            super.new(name);
        endfunction: new
        task body();
            seq_item = FIFO_seq_item::type_id::create("seq_item");
            repeat(100) begin
                seq_item.RD_EN_ON_DIST = read_dist;
                seq_item.WR_EN_ON_DIST = write_dist;
                start_item(seq_item);
                assert(seq_item.randomize());
                finish_item(seq_item);
            end
        endtask
    endclass: FIFO_write_only_sequence
    class FIFO_read_only_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_read_only_sequence)
        FIFO_seq_item seq_item;
        int read_dist = 100 ; // No read
        int write_dist = 0 ; // Write only
        function new(string name = "FIFO_read_only_sequence");
            super.new(name);
        endfunction: new
        task body();
            seq_item = FIFO_seq_item::type_id::create("seq_item");
            repeat(100) begin
                seq_item.RD_EN_ON_DIST = read_dist;
                seq_item.WR_EN_ON_DIST = write_dist;
                start_item(seq_item);
                assert(seq_item.randomize());
                finish_item(seq_item);
            end
        endtask
```

```
    endclass: FIFO_read_only_sequence
    class FIFO_write_read_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_write_read_sequence)
        FIFO_seq_item seq_item;
        function new(string name = "FIFO_write_read_sequence");
            super.new(name);
        endfunction: new
        task body();
            seq_item = FIFO_seq_item::type_id::create("seq_item");
            repeat (9999) begin
                start_item(seq_item);
                assert(seq_item.randomize());
                finish_item(seq_item);
            end
        endtask
    endclass: FIFO_write_read_sequence
endpackage
```

m)Sequencer

```
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
endpackage
```

n) Test

```
package FIFO_test_pkg;
    import uvm_pkg::*;
    import FIFO_env_pkg::*;
    import FIFO_config_obj_pkg::*;
    import FIFO_sequence_pkg::*;
    `include "uvm_macros.svh"
    class FIFO_test extends uvm_test;
        `uvm_component_utils(FIFO_test)
        FIFO_env env;
        FIFO_config_obj FIFO_config_obj_test;
        FIFO_reset_sequence reset_seq;
```

```
        FIFO_write_only_sequence wr_seq;
        FIFO_read_only_sequence rd_seq;
        FIFO_write_read_sequence wr_rd_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_config_obj_test =
FIFO_config_obj::type_id::create("FIFO_config_obj_test");
            reset_seq = FIFO_reset_sequence::type_id::create("reset_seq");
            wr_seq = FIFO_write_only_sequence::type_id::create("wr_seq");
            rd_seq = FIFO_read_only_sequence::type_id::create("rd_seq");
            wr_rd_seq = FIFO_write_read_sequence::type_id::create("wr_rd_seq");
            // Retrieve the virtual interface from the uvm_config_db
            if(!uvm_config_db#(virtual FIFO_if)::get(this, "", "FIFO_IF",
FIFO_config_obj_test.FIFO_config_vif))
                `uvm_fatal("build_phase", "Test - Unable to get the virtual
interface of the FIFO from the uvm_config_db");
            // Set the virtual interface for all components under this test class
            uvm_config_db#(FIFO_config_obj)::set(this, "*", "CFG",
FIFO_config_obj_test);
        endfunction
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);
            // FIFO_1
            //reset sequence
            `uvm_info("run_phase", "Reset Asserted", UVM_LOW)
            reset_seq.start(env.agt.sqr);
            `uvm_info("run_phase", "Reset Deasserted", UVM_LOW)
            // FIFO_2
            // writing only sequence
            `uvm_info ("run_phase" , "Writing only Started ",UVM_LOW)
            wr_seq.start(env.agt.sqr);
            `uvm_info ("run_phase" , "Writing only Ended ",UVM_LOW)
            // FIFO_3
            // Reading only sequence
            `uvm_info ("run_phase" , "Reading only Started ",UVM_LOW)
            rd_seq.start(env.agt.sqr);
            `uvm_info ("run_phase" , "Reading only Ended ",UVM_LOW)
            // FIFO_4
            //main sequence
            `uvm_info("run_phase", "Write & Read Started", UVM_LOW)
            wr_rd_seq.start(env.agt.sqr);
```

```
            `uvm_info("run_phase", "Write & Read Ended", UVM_LOW)
            phase.drop_objection(this);
        endtask
    endclass
endpackage
```

o) Top

```
import uvm_pkg::*;
import FIFO_test_pkg::*;
`include "uvm_macros.svh"
module FIFO_top();
    bit clk;
    always #5 clk = ~clk;
    FIFO_if FIFOif (clk);
    FIFO dut (FIFOif);
    bind FIFO FIFO_SVA assert_inst (FIFOif);
    initial begin
        uvm_config_db #(virtual FIFO_if)::set(null, "uvm_test_top", "FIFO_IF",
FIFOif);
        run_test("FIFO_test");
    end
endmodule
```

8)Do File

src_files.list

```
FIFO.sv
FIFO_SVA.sv
FIFO_if.sv
FIFO_seq_item.sv
FIFO_sequencer.sv
FIFO_config_obj.sv
FIFO_monitor.sv
FIFO_driver.sv
FIFO_agent.sv
FIFO_coverage.sv
FIFO_scoreboard.sv
FIFO_env.sv
FIFO_sequence.sv
FIFO_test.sv
FIFO_top.sv
```

run.do

```
vlib work
vlog -f src_files.list +cover -covercells
vsim -voptargs=+acc work.FIFO_top -classdebug -uvmcontrol=all -cover
coverage save FIFO.ucdb -onexit
add wave /FIFO_top/FIFOif/*
run -all
```

9)Code Coverage

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         86        86         0   100.00%
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        25        25         0   100.00%
================================================================================
Condition Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Conditions                      22        22         0   100.00%
================================================================================
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      19        19         0   100.00%
================================================================================
```

10) Functional Coverage

Covergroups

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

11) Assertion Coverage

Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Pea
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1...	Immediate	SVA	on	0	0	-	-	-	
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1...	Immediate	SVA	on	0	0	-	-	-	
/FIFO_sequence_pkg::FIFO_write_only_sequence::body/#ublk#386150...	Immediate	SVA	on	0	1	-	-	-	
/FIFO_sequence_pkg::FIFO_read_only_sequence::body/#ublk#386150...	Immediate	SVA	on	0	1	-	-	-	
/FIFO_sequence_pkg::FIFO_write_read_sequence::body/#ublk#386150...	Immediate	SVA	on	0	1	-	-	-	
/FIFO_top/dut/assert_inst/wr_count_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/FIFO_top/dut/assert_inst/rd_count_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/FIFO_top/dut/assert_inst/rd_wr_count_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/FIFO_top/dut/assert_inst/rd_wr_count_empty_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/FIFO_top/dut/assert_inst/rd_wr_count_full_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/FIFO_top/dut/assert_inst/wr_ptr_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/FIFO_top/dut/assert_inst/rd_ptr_assert	Concurrent	SVA	on	0	1	-	0B	0B	
/FIFO_top/dut/assert_inst/rst_count_assert	Immediate	SVA	on	0	1	-	-	-	
/FIFO_top/dut/assert_inst/rst_wr_ptr_assert	Immediate	SVA	on	0	1	-	-	-	
/FIFO_top/dut/assert_inst/rst_rd_ptr_assert	Immediate	SVA	on	0	1	-	-	-	
/FIFO_top/dut/assert_inst/full_assert	Immediate	SVA	on	0	1	-	-	-	
/FIFO_top/dut/assert_inst/empty_assert	Immediate	SVA	on	0	1	-	-	-	
/FIFO_top/dut/assert_inst/underflow_assert	Immediate	SVA	on	0	1	-	-	-	
/FIFO_top/dut/assert_inst/overflow_assert	Immediate	SVA	on	0	1	-	-	-	
/FIFO_top/dut/assert_inst/almostfull_assert	Immediate	SVA	on	0	1	-	-	-	
/FIFO_top/dut/assert_inst/almostempty_assert	Immediate	SVA	on	0	1	-	-	-	
/FIFO_top/dut/assert_inst/wr_ack_assert	Immediate	SVA	on	0	1	-	-	-	

Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Mer
/FIFO_top/dut/assert_inst/wr_count_c...	SVA	✓	Off	3598	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/rd_count_c...	SVA	✓	Off	583	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/rd_wr_coun...	SVA	✓	Off	1286	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/rd_wr_coun...	SVA	✓	Off	241	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/rd_wr_coun...	SVA	✓	Off	161	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/wr_ptr_cov...	SVA	✓	Off	5125	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/rd_ptr_cove...	SVA	✓	Off	2030	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/rst_count_c...	SVA	✓	Off	1000	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/rst_wr_ptr_...	SVA	✓	Off	1000	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/rst_rd_ptr_...	SVA	✓	Off	1000	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/full_cover	SVA	✓	Off	886	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/empty_cove...	SVA	✓	Off	2470	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/underflow_c...	SVA	✓	Off	829	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/overflow_co...	SVA	✓	Off	643	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/almostfull_c...	SVA	✓	Off	1326	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/almostempt...	SVA	✓	Off	2555	1	Unlimited	1	100%		✓	0	0	
/FIFO_top/dut/assert_inst/wr_ack_cov...	SVA	✓	Off	9969	1	Unlimited	1	100%		✓	0	0	

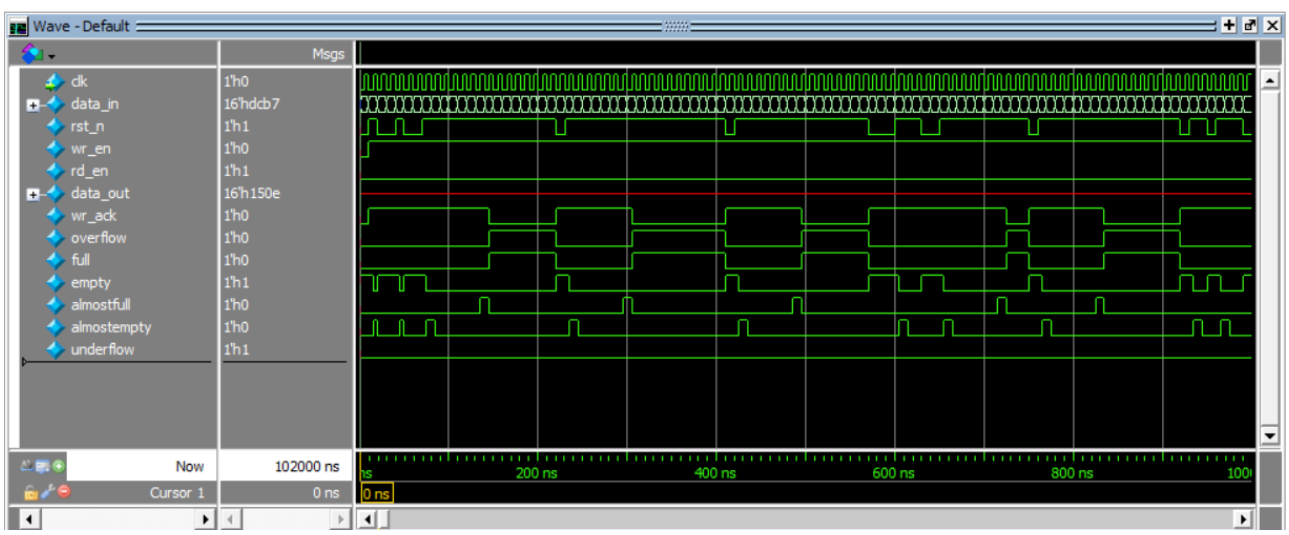
12) Questa Sim Snippets

a) Transcript

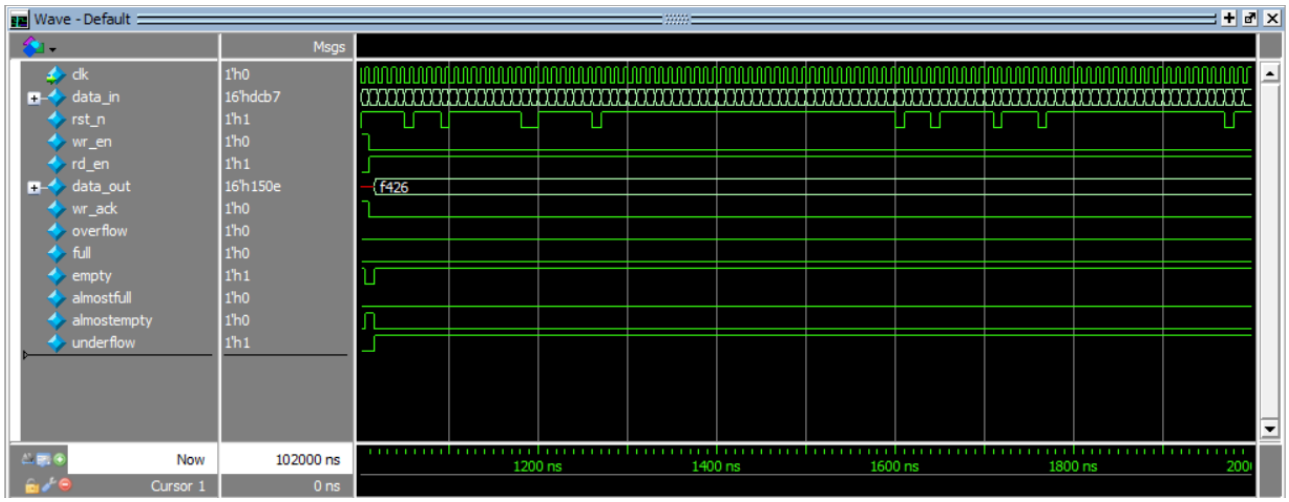
```
Transcript
File  Edit  View  Bookmarks  Window  Help
Transcript
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
#   See http://www.eda.org/svdb/view.php?id=3770 for more details.
#
#       (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM]  questa_uvm::init(all)
# UVM_INFO @ 0: reporter [RNTST] Running test FIFO_test...
# UVM_INFO FIFO_test.sv(46) @ 0: uvm_test_top [run_phase] Reset Asserted
# ************************************************************************
# * Questa UVM Transaction Recording Turned ON.                          *
# * recording_detail has been set.                                       *
# *  To turn off, set 'recording_detail' to off:                         *
# * uvm_config_db#(int)            ::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0); *
# ************************************************************************
# UVM_INFO FIFO_test.sv(48) @ 10: uvm_test_top [run_phase] Reset Deasserted
# UVM_INFO FIFO_test.sv(52) @ 10: uvm_test_top [run_phase] Writing only Started
# UVM_INFO FIFO_test.sv(54) @ 1010: uvm_test_top [run_phase] Writing only Ended
# UVM_INFO FIFO_test.sv(58) @ 1010: uvm_test_top [run_phase] Reading only Started
# UVM_INFO FIFO_test.sv(60) @ 2010: uvm_test_top [run_phase] Reading only Ended
# UVM_INFO FIFO_test.sv(64) @ 2010: uvm_test_top [run_phase] Write & Read Started
# UVM_INFO FIFO_test.sv(66) @ 102000: uvm_test_top [run_phase] Write & Read Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 102000: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO FIFO_scoreboard.sv(94) @ 102000: uvm_test_top.env.sb [report_phase] Total successful transactions: 10200
# UVM_INFO FIFO_scoreboard.sv(95) @ 102000: uvm_test_top.env.sb [report_phase] Total failed transactions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   14
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# [run_phase]     8
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 102 us  Iteration: 61  Instance: /FIFO_top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430

VSIM 12>
```

b) Write Only Waveform



c) Read Only Waveform



d) Write & Read Waveform