Synchronous FIFO – UVM

Amira Atef Ismaeil El Komy

Table of Contents

List of Figures

List of Tables

1. Verification Plan

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	When the reset_n is asserted, the output count_out value should be low, max_count should be low and zero should be high.	Directed at the start of the simulation, then randomized with constraint that drive the reset to be off(high) most of the simulation time.	-	Immediate assertion to check for the async reset functionality.
FIFO_2	When the rd_en is asserted and wr_en is de-asserted, the output data_out should take the value of the stored current value.	Randomization under constraints on rd_en being on for RD_EN_ON_DISTANCE%.	-	Checked by the reference model. Concurrent assertion to check for the counter decrement.
FIFO_3	When the wr_en is asserted and rd_en is de-asserted, the input data_in should be stored in the fifo, and the output wr_ack should get asserted.	Cyclic randomization on the data_in. Randomization under constraints on wr_en being on for WR_EN_ON_DISTANCE% of the time.	Cover all values of data_in. Cover the crossing of the wr_ack signal with rd_en and wr_en.	Concurrent assertion to check for the wr_ack signal, can also be checked by the reference model. Concurrent assertion to check for the counter increment.
FIFO_4	When the wr_en and rd_en are asserted without empty or full being asserted, the input data_in should be stored in the fifo, and the output wr_ack should get asserted and the current stored value to be read will be output on data_out.	Randomization under constraints on the ce signal to be on(high) 70% of the time.	-	Checked by the reference model. Concurrent assertion to check for the wr_ack signal, can also be checked by the reference model. Concurrent assertion to check for the counter remaining unchanged.
FIFO_5	When the wr_en and rd_en are asserted with empty being asserted, the input data_in should be stored in the fifo, and the output wr_ack should get asserted.	Randomization	Cover the crossing of the wr_ack signal with rd_en and wr_en.	Concurrent assertion to check for the wr_ack signal, can also be checked by the reference model. Concurrent assertion to check for the counter increment.
FIFO_6	When the wr_en and rd_en are asserted with full being asserted, the current stored value to be read will be output on data_out.	Randomization	-	Checked by the reference model. Concurrent assertion to check for the counter decrement.
FIFO_7	When the fifo is full, output flag full must be asserted.	Randomization	Cover the crossing of the full signal with rd_en and wr_en.	Concurrent assertion to check for the full signal, can also be checked by the reference model.
FIFO_8	When the fifo is full and wr_en gets asserted, output flag overflow must be asserted.	Randomization	Cover the crossing of the overflow signal with rd_en and wr_en.	Concurrent assertion to check for the overflow signal, can also be checked by the reference model.
FIFO_9	When the fifo is empty, output flag empty must be asserted.	Randomization	Cover the crossing of the empty signal with rd_en and wr_en.	Concurrent assertion to check for the empty signal, can also be checked by the reference model.
FIFO_10	When the fifo is empty and wr_en gets asserted, output flag underflow must be asserted.	Randomization	Cover the crossing of the underflow signal with rd_en and wr_en.	Concurrent assertion to check for the underflow signal, can also be checked by the reference model.
FIFO_11	When the fifo has only 1 data value stored, output flag almostempty must be asserted.	Randomization	Cover the crossing of the almostempty signal with rd_en and wr_en.	Concurrent assertion to check for the almostempty signal, can also be checked by the reference model.
FIFO_12	When the fifo has 1 remaining spot left to store, output flag almostfull must be asserted.	Randomization	Cover the crossing of the almostfull signal with rd_en and wr_en.	Concurrent assertion to check for the almostfull signal, can also be checked by the reference model.

Figure 1: Sync. FIFO Verification Plan

2. UVM Testbench Structure

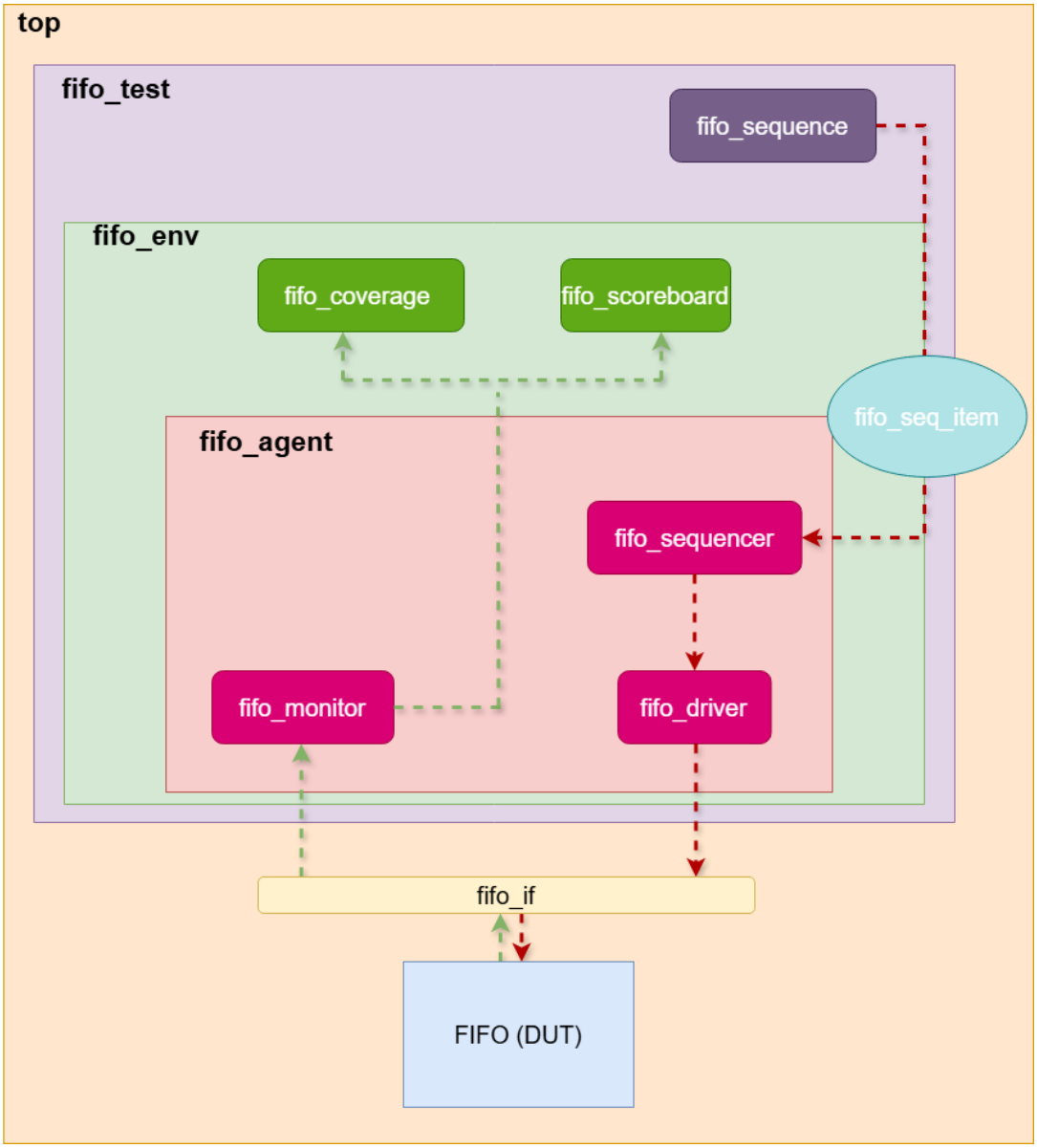

Figure 2: FIFO UVM Environment Architecture

The UVM testbench flow:

- The top module generates the clock and instantiates the test, interface and FIFO, our design under test.
- The interface supplies the DUT with the input stimulus and receives the output from it, so the interface carries all the signals.
- The test class, which extends uvm_test, instantiates all sequence classes and the environment. It also runs all the sequences.

- The sequence item class, which extends uvm_seq_item, contains all the variables for all the signals passing through the environment whether these signals are DUT input signals that get randomized or not. In addition, it contains all constraint blocks for randomized signals.
- The sequence class, which extends uvm_sequence, is a base class that contains tasks for all different sequences, which are then called in child classes of the base class to run different sequences.
- The environment class, which extends uvm_env, instantiates the coverage collector, scoreboard and the agent.
- The agent class, which extends uvm_agent, instantiates the monitor, driver and sequencer, as well as creates the connections using uvm_tlm_ports and uvm_analysis_ports, between the monitor and classes outside the agent or between the driver and sequencer, respectively.
- The driver class, which extends uvm_driver, instantiates a seq_item, as it drives the input stimulus to the interface given to it by the sequence in a seq_item through the sequencer.
- The monitor class, which extends uvm_monitor, instantiates a seq_item, and writes all the signals in the interface to a uvm_fifo every negative edge clock.
- The coverage class, which extends uvm_component, instantiates a seq_item, to be able to get the sequence item from the monitor and applies coverpoints inside covergroups to its signals and samples the covergroups.
- The scoreboard class, which extends uvm_scoreboard, instantiates a seq_item, and compares every output using a reference model task written inside it, incrementing counters for every correct and incorrect comparison.
- An SVA class was also written containing all assertions and was bound to the DUT inside the top.

3. Bugs Found

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        wr_ptr <= 0;
        wr_ack <= 0;   //wr_ack and overflow were not reset
        overflow <= 0;
    end
    else if (wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= data_in;
        wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
        overflow <= 0;  //overflow should be zero if wr_en and !full
```

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        wr_ptr <= 0;
    end
    else if (wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= data_in;
        wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    end
    else begin
        wr_ack <= 0;
```

Figure 3: Bugs Found 1

```
    else begin
        wr_ack <= 0;
        if (full && wr_en) begin // should be && not & for conditional coverage
            overflow <= 1;
        end
        else begin
            overflow <= 0;
        end
```

```
        wr_ack <= 0;
        if (full & wr_en)
            overflow <= 1;
        else
            overflow <= 0;
    end
end
```

Figure 4: Bugs Found 2

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rd_ptr <= 0;
        data_out <= 0;  //data_out was not reset
        underflow <= 0; // rst_n was not accounted for
    end
    else if (rd_en && count != 0) begin
        data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        underflow <= 1'b0;
    end
    else if (rd_en) begin
        underflow <= 1'b1;
    end
    else begin
        underflow <= 1'b0;
    end
end
```

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rd_ptr <= 0;
    end
    else if (rd_en && count != 0) begin
        data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
end
```

Figure 5: Bugs Found 3

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        count <= 0;
    end
    else begin
        if ({wr_en, rd_en} == 2'b11) begin  //The possibility of rd & wr enable with eithe
            if (empty)
                count <= count + 1;
            else if (full)
                count <= count - 1;
        end
        else if ( ({wr_en, rd_en} == 2'b10) && !full)
            count <= count + 1;
        else if (rd_en && !empty) // for conditional coverage
            count <= count - 1;
    end
end
```

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({wr_en, rd_en} == 2'b10) && !full)
            count <= count + 1;
        else if ( ({wr_en, rd_en} == 2'b01) && !empty)
            count <= count - 1;
    end
end
```

Figure 6: Bugs Found 4

```
assign full = (count == FIFO_DEPTH)? 1 : 0;
assign empty = (count == 0)? 1 : 0; // underflow was placed inside read always block
assign almostfull = (count == FIFO_DEPTH-1)? 1 : 0; // it was FIFO_DEPTH-2, almostfull mea
assign almostempty = (count == 1)? 1 : 0;
```

```
assign full = (count == FIFO_DEPTH)? 1 : 0;
assign empty = (count == 0)? 1 : 0;
assign underflow = (empty && rd_en)? 1 : 0;
assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
assign almostempty = (count == 1)? 1 : 0;
```

Figure 7: Bugs Found 5

4. Source Files List

```
src_files.list
shared_pkg.sv
fifo_transactions_pkg.sv
fifo_agent_pkg.sv
fifo_env_pkg.sv
fifo_test_pkg.sv
fifo_if.sv
FIFO.sv
fifo_sva.sv
top.sv
```

Figure 8: List of Source Files

5. Do File

```
RUN.do
vlib work
vlog -f src_files.list +cover -covercells
vsim -voptargs=+acc work.top -cover
add wave /top/fifoif/*
coverage save FIFO_UVM.ucdb -onexit -du work.FIFO
vcover report FIFO_UVM.ucdb -details -annotate -all -output FIFO_UVM_cvr_rpt.txt
run -all
```

Figure 9: The Do File

6. Coverage Report Snippets

5.1. Branch Coverage Snippet

```
Branches - by instance (/top/DUT)
FIFO.sv
 39 if (!rst_n) begin
 44 else if (wr_en && count < FIFO_DEPTH) begin
 50 else begin
 52 if (full && wr_en) begin // should be && not & for conditional coverage
 55 else begin
 62 if (!rst_n) begin
 67 else if (rd_en && count != 0) begin
 72 else if (rd_en) begin
 75 else begin
 81 if (!rst_n) begin
 84 else begin
 85 if ({wr_en, rd_en} == 2'b11) begin  //The possibility of rd & wr enable with either empty or full was not considered.
 86 if (empty)
 88 else if       (full)
 91 else if       ( ({wr_en, rd_en} == 2'b10) && !full)
 93 else if (rd_en && !empty) // for conditional coverage
 98 assign full = (count == FIFO_DEPTH)? 1 : 0;
 99 assign empty = (count == 0)? 1 : 0; // underflow was placed inside read always block
100 assign almostfull = (count == FIFO_DEPTH-1)? 1 : 0; // it was FIFO_DEPTH-2, almostfull means 1 element can be added
101 assign almostempty = (count == 1)? 1 : 0;
top.sv
```

Figure 10: Total Branch Coverage

5.2. Toggle Coverage Snippet

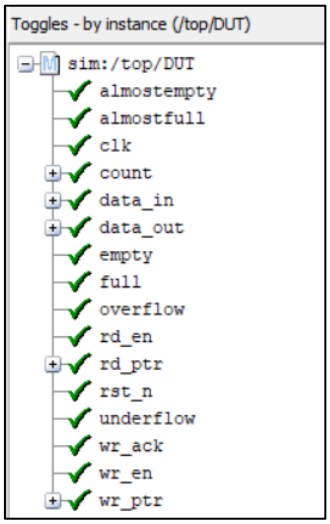

Figure 11: Total Toggle Coverage

5.3. Statement Coverage Snippet

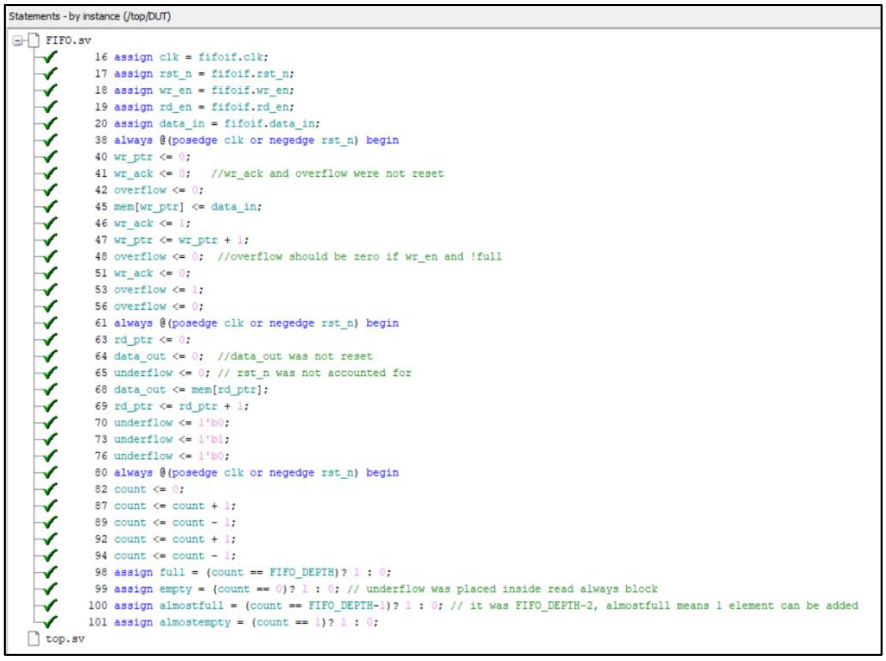

Figure 12: Total Statement Coverage

5.4. Condition Coverage Snippet

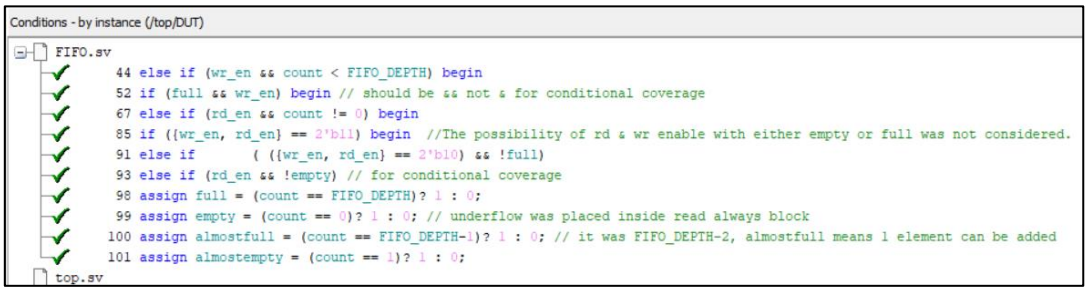

Figure 13: Total Condition Coverage

7. Functional Coverage Report Snippet

Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included
/fifo_env_pkg/fifo_coverage		100.00%				
TYPE FIFO_cvr_grp		100.00%	100	100.00...		✓
CVP FIFO_cvr_grp::rst_n_c		100.00%	100	100.00...		✓
CVP FIFO_cvr_grp::wr_en_c		100.00%	100	100.00...		✓
CVP FIFO_cvr_grp::rd_en_c		100.00%	100	100.00...		✓
CVP FIFO_cvr_grp::wr_ack_c		100.00%	100	100.00...		✓
CVP FIFO_cvr_grp::overflow_c		100.00%	100	100.00...		✓
CVP FIFO_cvr_grp::full_c		100.00%	100	100.00...		✓
CVP FIFO_cvr_grp::empty_c		100.00%	100	100.00...		✓
CVP FIFO_cvr_grp::almostfull_c		100.00%	100	100.00...		✓
CVP FIFO_cvr_grp::almostempty_c		100.00%	100	100.00...		✓
CVP FIFO_cvr_grp::underflow_c		100.00%	100	100.00...		✓
CROSS FIFO_cvr_grp::wr_ack_cross		100.00%	100	100.00...		✓
CROSS FIFO_cvr_grp::overflow_cross		100.00%	100	100.00...		✓
CROSS FIFO_cvr_grp::full_cross		100.00%	100	100.00...		✓
CROSS FIFO_cvr_grp::empty_cross		100.00%	100	100.00...		✓
CROSS FIFO_cvr_grp::almostfull_cross		100.00%	100	100.00...		✓
CROSS FIFO_cvr_grp::almostempty_cross		100.00%	100	100.00...		✓
CROSS FIFO_cvr_grp::underflow_cross		100.00%	100	100.00...		✓

Figure 14: Total Functional Coverage

8. Assertions Report Snippet

```
Coverage Report by instance with details

=======================================================================
=== Instance: /top/DUT/SVA
=== Design Unit: work.fifo_sva
=======================================================================

Assertion Coverage:
    Assertions                      10        10         0   100.00%
----------------------------------------------------------------------
Name                File(Line)                    Failure      Pass
                                                  Count        Count
----------------------------------------------------------------------
/top/DUT/SVA/assert__count_rd_prop
                    fifo_sva.sv(96)                    0          1
/top/DUT/SVA/assert__count_wr_prop
                    fifo_sva.sv(95)                    0          1
/top/DUT/SVA/assert__almostempty_prop
                    fifo_sva.sv(94)                    0          1
/top/DUT/SVA/assert__almostfull_prop
                    fifo_sva.sv(93)                    0          1
/top/DUT/SVA/assert__empty_prop
                    fifo_sva.sv(92)                    0          1
/top/DUT/SVA/assert__full_prop
                    fifo_sva.sv(91)                    0          1
/top/DUT/SVA/assert__wr_ack_prop
                    fifo_sva.sv(90)                    0          1
/top/DUT/SVA/assert__underflow_prop
                    fifo_sva.sv(89)                    0          1
/top/DUT/SVA/assert__overflow_prop
                    fifo_sva.sv(88)                    0          1
/top/DUT/SVA/#ublk#265645057#46/immed__47
                    fifo_sva.sv(47)                    0          1
```

Figure 15: Assertions

Table 1: Assertions

Feature	Assertion
When rst_n is zero all outputs must be zero and only empty is 1.	always @(posedge clk) begin if (!rst_deasserted) **assert** (data_out == 0 && empty && !almostfull && !almostempty && !underflow && !overflow && !full && !wr_ack && count == 0) end
When FIFO is full and wr_en gets asserted, overflow gets asserted.	@(posedge clk) disable iff (~rst_n) (wr_en && full) |=> overflow;
When FIFO is empty and rd_en gets asserted, overflow gets asserted.	@(posedge clk) disable iff (~rst_n) (rd_en && empty) |=> underflow;
When FIFO is not full and wr_en gets asserted, wr_ack gets asserted.	@(posedge clk) disable iff (~rst_n) (wr_en && count < FIFO_DEPTH) |=> wr_ack;
When FIFO is full, as indicated by the counter, Full gets asserted.	@(posedge clk) disable iff (~rst_n) (count == FIFO_DEPTH) |-> full;
When FIFO is empty, as indicated by the counter, empty gets asserted.	@(posedge clk) disable iff (~rst_n) (count == 0) |-> empty;
When only 1 place is empty in FIFO, almostfull gets asserted.	@(posedge clk) disable iff (~rst_n) (count == FIFO_DEPTH-1) |-> almostfull;
When only 1 place is full in FIFO, almostempty gets asserted.	@(posedge clk) disable iff (~rst_n) (count == 1) |-> almostempty;
Counter must get incremented when wr_en is asserted, fifo is not full and rd_en is de-asserted or when wr_en is asserted, fifo is empty and rd_en is asserted.	@(posedge clk) disable iff (~rst_n) ((wr_en && !full && !rd_en) || (wr_en && rd_en && empty)) |=> (count == **$past**(count) + 1);
Counter must get decremented when rd_en is asserted, fifo is not empty and wr_en is de-asserted or when rd_en is asserted, fifo is full and wr_en is asserted.	@(posedge clk) disable iff (~rst_n) ((rd_en && !empty && !wr_en) || (wr_en && rd_en && full)) |=> (count == **$past**(count) - 1);

9. Cover Directives Snippet

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/top/DUT/SVA/cover__count_rd_pr...	SVA	✓	Off	23	1	Unli...	1	100%		✓	0	0	0 ps	0
/top/DUT/SVA/cover__count_wr_pr...	SVA	✓	Off	83	1	Unli...	1	100%		✓	0	0	0 ps	0
/top/DUT/SVA/cover__almostempty...	SVA	✓	Off	24	1	Unli...	1	100%		✓	0	0	0 ps	0
/top/DUT/SVA/cover__almostfull_pr...	SVA	✓	Off	25	1	Unli...	1	100%		✓	0	0	0 ps	0
/top/DUT/SVA/cover__empty_prop	SVA	✓	Off	56	1	Unli...	1	100%		✓	0	0	0 ps	0
/top/DUT/SVA/cover__full_prop	SVA	✓	Off	67	1	Unli...	1	100%		✓	0	0	0 ps	0
/top/DUT/SVA/cover__wr_ack_prop...	SVA	✓	Off	108	1	Unli...	1	100%		✓	0	0	0 ps	0
/top/DUT/SVA/cover__underflow_p...	SVA	✓	Off	30	1	Unli...	1	100%		✓	0	0	0 ps	0
/top/DUT/SVA/cover__overflow_pr...	SVA	✓	Off	59	1	Unli...	1	100%		✓	0	0	0 ps	0

Figure 16: Cover Directives

10. Waveform Snippets

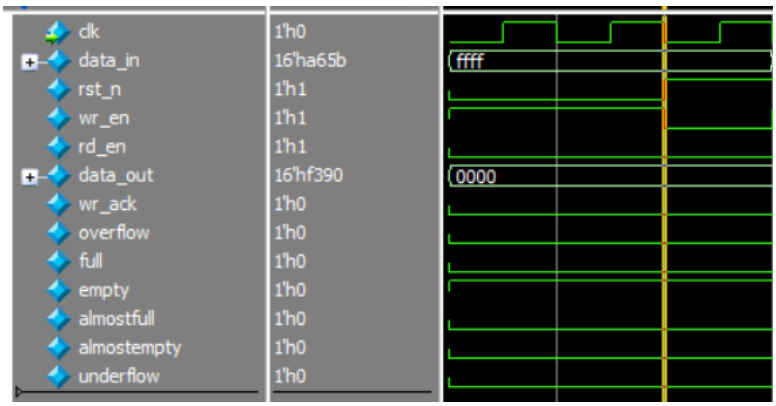

Figure 17: Reset Sequence Waveform

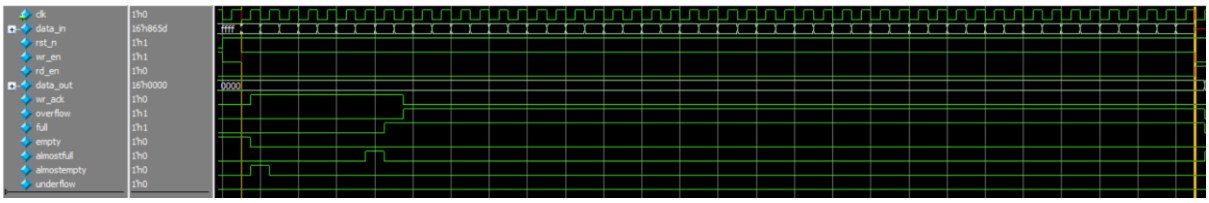

Figure 18: Write Only Sequence Waveform

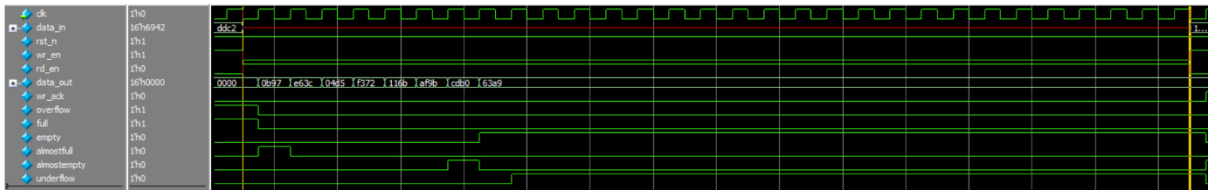

Figure 19: Read Only Sequence Waveform

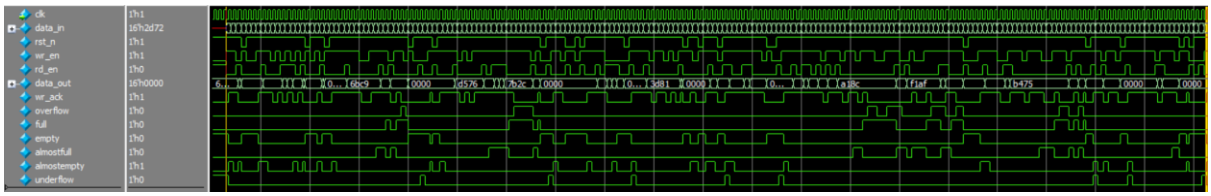

Figure 20: Write + Read Sequence Waveform

11. Transcript Snippets

```
# ----------------------------------------------------------------
# UVM-1.1d
# (C) 2007-2013 Mentor Graphics Corporation
# (C) 2007-2013 Cadence Design Systems, Inc.
# (C) 2006-2013 Synopsys, Inc.
# (C) 2011-2013 Cypress Semiconductor Corp.
# ----------------------------------------------------------------
#
#   ***********       IMPORTANT RELEASE NOTES         ************
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_NO_DEPRECATED undefined.
#   See http://www.eda.org/svdb/view.php?id=3313 for more details.
#
#   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(+struct)
# UVM_INFO @ 0: reporter [RNTST] Running test fifo_test...
# UVM_INFO fifo_test.svh(48) @ 0: uvm_test_top [run_phase] RESET stimulus generation started
# UVM_INFO fifo_scoreboard.svh(41) @ 0: uvm_test_top.env.sb [fifo_scoreboard] Entered scoreboard run_phase
# UVM_INFO reset_sequence.svh(16) @ 0: uvm_test_top.env.agt.sqr@@reset_seq [reset_sequence] Starting reset sequence
# UVM_INFO fifo_base_sequence.svh(21) @ 0: uvm_test_top.env.agt.sqr@@reset_seq [reset_sequence] Resetting DUT
# UVM_INFO fifo_base_sequence.svh(37) @ 20000: uvm_test_top.env.agt.sqr@@reset_seq [reset_sequence] DUT Reset complete
# UVM_INFO reset_sequence.svh(20) @ 30000: uvm_test_top.env.agt.sqr@@reset_seq [reset_sequence] Completed reset sequence
# UVM_INFO fifo_test.svh(50) @ 30000: uvm_test_top [run_phase] RESET generation ended
# UVM_INFO fifo_test.svh(52) @ 30000: uvm_test_top [run_phase] WRITE ONLY stimulus generation started
# UVM_INFO write_only_sequence.svh(16) @ 30000: uvm_test_top.env.agt.sqr@@wr_seq [write_only_sequence] Starting write only sequence
# UVM_INFO fifo_base_sequence.svh(48) @ 30000: uvm_test_top.env.agt.sqr@@wr_seq [write_only_sequence] start write only task
# UVM_INFO fifo_base_sequence.svh(62) @ 530000: uvm_test_top.env.agt.sqr@@wr_seq [write_only_sequence] Completed write only task
# UVM_INFO write_only_sequence.svh(20) @ 530000: uvm_test_top.env.agt.sqr@@wr_seq [write_only_sequence] Completed write only sequence
# UVM_INFO fifo_test.svh(54) @ 530000: uvm_test_top [run_phase] WRITE ONLY stimulus generation ended
# UVM_INFO fifo_test.svh(56) @ 530000: uvm_test_top [run_phase] READ ONLY stimulus generation started
# UVM_INFO read_only_sequence.svh(16) @ 530000: uvm_test_top.env.agt.sqr@@rd_seq [read_only_sequence] Starting read only sequence
# UVM_INFO fifo_base_sequence.svh(68) @ 530000: uvm_test_top.env.agt.sqr@@rd_seq [read_only_sequence] start read only task
# UVM_INFO fifo_base_sequence.svh(84) @ 830000: uvm_test_top.env.agt.sqr@@rd_seq [read_only_sequence] Completed read only task
# UVM_INFO read_only_sequence.svh(20) @ 830000: uvm_test_top.env.agt.sqr@@rd_seq [read_only_sequence] Completed read only sequence
# UVM_INFO fifo_test.svh(58) @ 830000: uvm_test_top [run_phase] READ ONLY stimulus generation ended
# UVM_INFO fifo_test.svh(60) @ 830000: uvm_test_top [run_phase] WRITE + READ stimulus generation started
# UVM_INFO write_read_sequence.svh(16) @ 830000: uvm_test_top.env.agt.sqr@@wr_rd_seq [write_read_sequence] Starting write + read sequence
# UVM_INFO fifo_base_sequence.svh(90) @ 830000: uvm_test_top.env.agt.sqr@@wr_rd_seq [write_read_sequence] start write + read task
# UVM_INFO fifo_base_sequence.svh(106) @ 2830000: uvm_test_top.env.agt.sqr@@wr_rd_seq [write_read_sequence] Completed write + read task
# UVM_INFO write_read_sequence.svh(20) @ 2830000: uvm_test_top.env.agt.sqr@@wr_rd_seq [write_read_sequence] Completed write + read sequence
```

Figure 21: Transcript Snippet

```
# UVM_INFO fifo_test.svh(62) @ 2830000: uvm_test_top [run_phase] WRITE + READ stimulus generation ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 2830000: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO fifo_scoreboard.svh(168) @ 2830000: uvm_test_top.env.sb [report_phase] Total successful transactions: 2264.
# UVM_INFO fifo_scoreboard.svh(169) @ 2830000: uvm_test_top.env.sb [report_phase] Total failed transactions: 0.
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   31
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [fifo_scoreboard]     1
# [read_only_sequence]     4
# [report_phase]     2
# [reset_sequence]     4
# [run_phase]     8
# [write_only_sequence]     4
# [write_read_sequence]     4
# ** Note: $finish    : D:/qsim_2021_1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 2830 ns  Iteration: 61  Instance: /top
```

Figure 22: Transcript Snippet (Cont.)