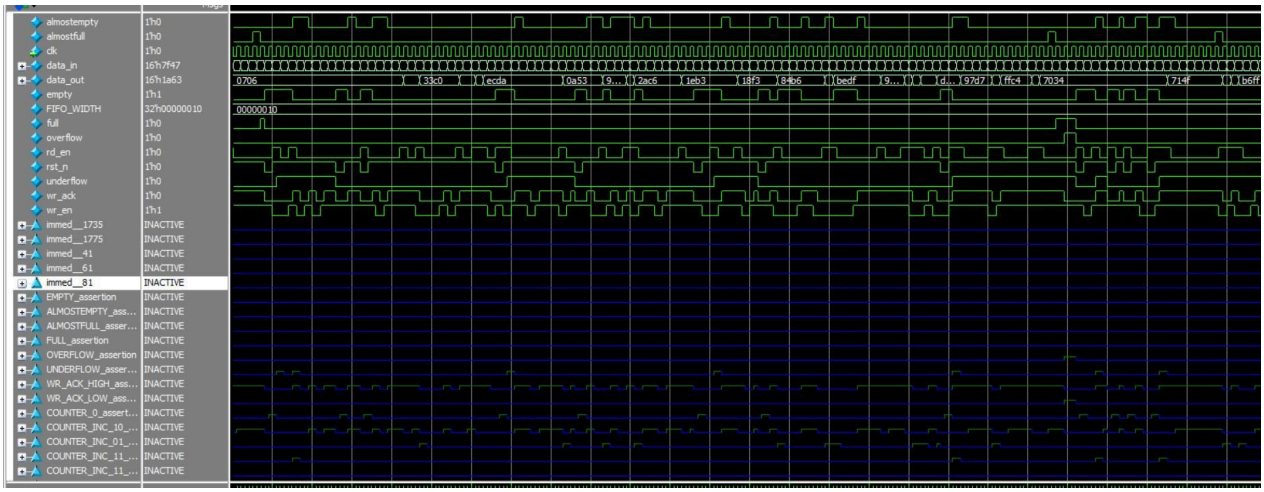
UVM Project

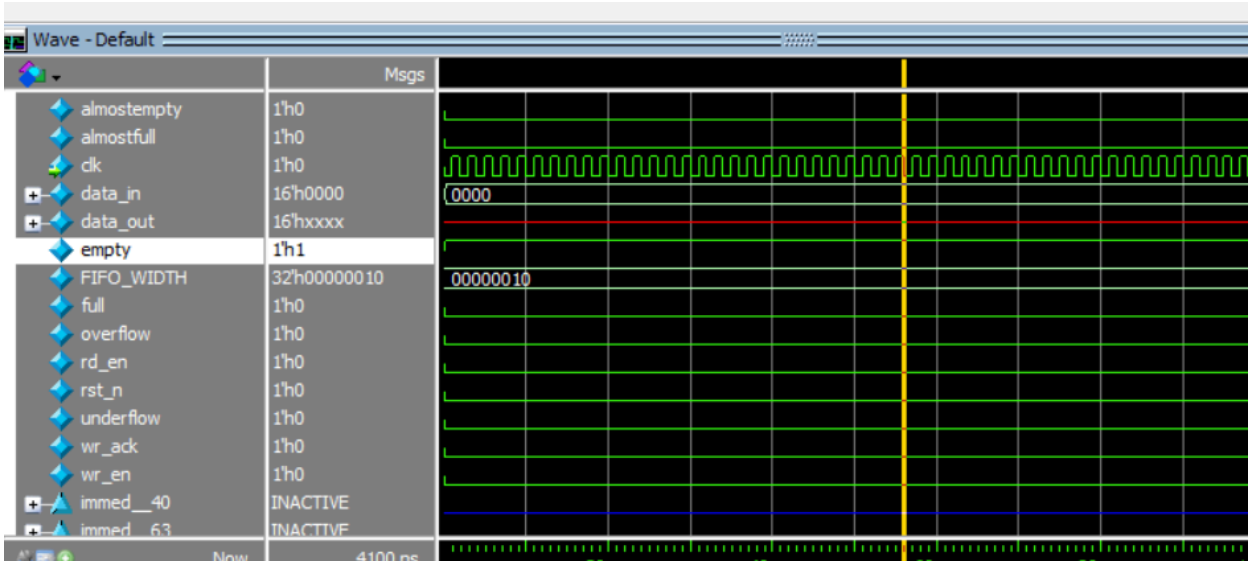
Questasim snippets:



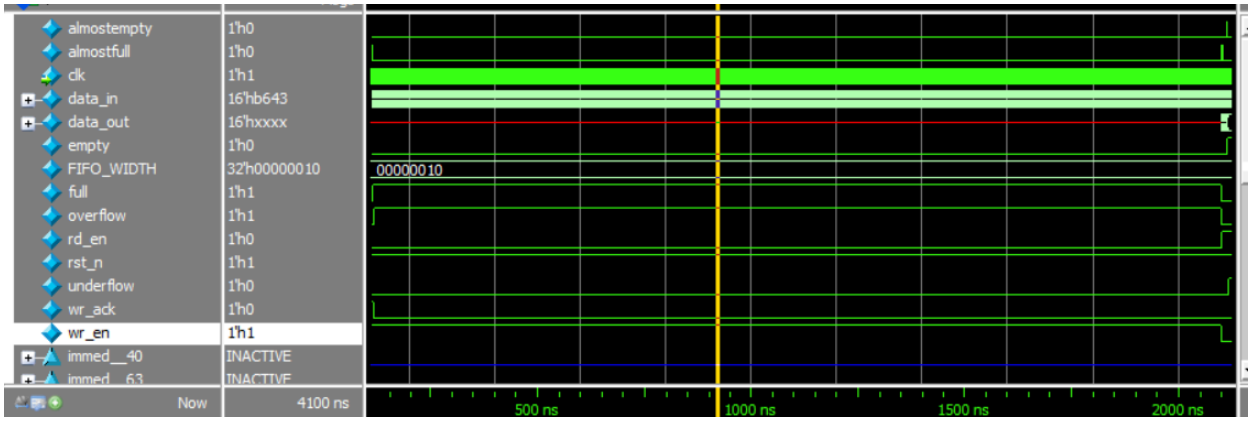

```
# UVM_INFO FIFO_test.sv(42) @ 4: uvm_test_top [run_phase] Ending FIFO reset sequence
# UVM_INFO FIFO_test.sv(43) @ 4: uvm_test_top [run_phase] Starting write only sequence
# UVM_INFO FIFO_test.sv(45) @ 2004: uvm_test_top [run_phase] Ending write only sequence
# UVM_INFO FIFO_test.sv(46) @ 2004: uvm_test_top [run_phase] Starting read only sequence
# UVM_INFO FIFO_test.sv(48) @ 3004: uvm_test_top [run_phase] Ending read only sequence
# UVM_INFO FIFO_test.sv(49) @ 3004: uvm_test_top [run_phase] Starting read write sequence
# UVM_INFO FIFO_test.sv(51) @ 4004: uvm_test_top [run_phase] Ending read write sequence
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 4004: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO FIFO_Scoreboard.sv(128) @ 4004: uvm_test_top.env.sb [report_phase] Total correct:2002
# UVM_INFO FIFO_Scoreboard.sv(129) @ 4004: uvm_test_top.env.sb [report_phase] Total error: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:/QuestaFolder/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
```

/sequence_fifo_pkg::write_only_sequence::body...	Immediate	SVA	on	0	1	-	-	-
/sequence_fifo_pkg::read_only_sequence::body...	Immediate	SVA	on	0	1	-	-	-
/sequence_fifo_pkg::read_write_sequence::bod...	Immediate	SVA	on	0	1	-	-	-
/top/DUTf/sva1/EMPTY_assertion	Immediate	SVA	on	0	1	-	-	-
/top/DUTf/sva1/ALMOSTEMPTY_assertion	Immediate	SVA	on	0	1	-	-	-
/top/DUTf/sva1/ALMOSTFULL_assertion	Immediate	SVA	on	0	1	-	-	-
/top/DUTf/sva1/FULL_assertion	Immediate	SVA	on	0	1	-	-	-
/top/DUTf/sva1/OVERFLOW_assertion	Concurrent	SVA	on	0	1	-	0B	0B
/top/DUTf/sva1/UNDERFLOW_assertion	Concurrent	SVA	on	0	1	-	0B	0B
/top/DUTf/sva1/WR_ACK_HIGH_assertion	Concurrent	SVA	on	0	1	-	0B	0B
/top/DUTf/sva1/WR_ACK_LOW_assertion	Concurrent	SVA	on	0	1	-	0B	0B
/top/DUTf/sva1/COUNTER_0_assertion	Concurrent	SVA	on	0	1	-	0B	0B
/top/DUTf/sva1/COUNTER_INC_10_assertion	Concurrent	SVA	on	0	1	-	0B	0B
/top/DUTf/sva1/COUNTER_INC_01_assertion	Concurrent	SVA	on	0	1	-	0B	0B
/top/DUTf/sva1/COUNTER_INC_11_WR_assertio...	Concurrent	SVA	on	0	1	-	0B	0B
/top/DUTf/sva1/COUNTER_INC_11_RD_assertio...	Concurrent	SVA	on	0	1	-	0B	0B
/top/DUTf/sva1/COUNTER_LAT_assertion	Concurrent	SVA	on	0	1	-	0B	0B
/top/DUTf/sva1/PTR_RST_assertion	Concurrent	SVA	on	0	1	-	0B	0B
/top/DUTf/sva1/RD_PTR_assertion	Concurrent	SVA	on	0	1	-	0B	0B
/top/DUTf/sva1/WR_PTR_assertion	Concurrent	SVA	on	0	1	-	0B	0B

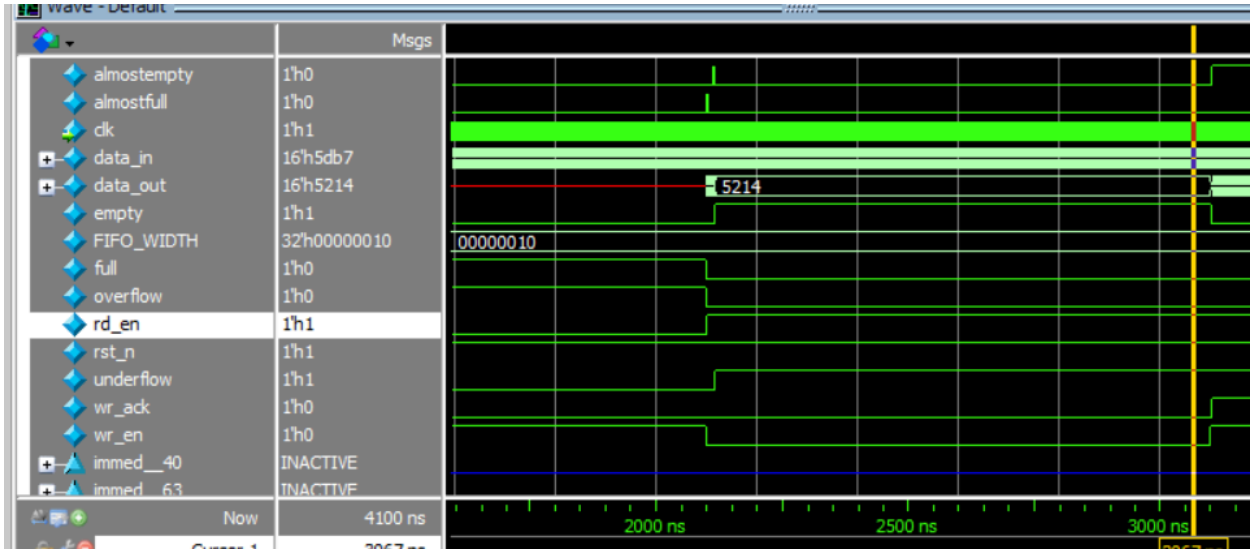
Reset Sequence:



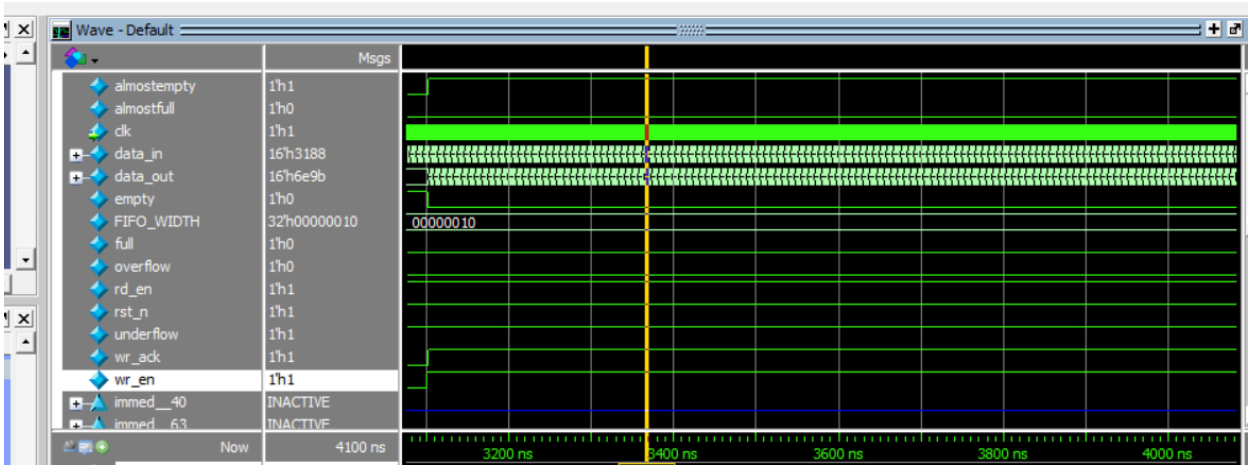
Write_only Sequence:



Read_only Sequence:



Read_Write_Sequence:



Sequential Domain Coverage report

```
=== Instance: /top/DUTf/sva1
=== Design Unit: work.SVA
================================================================================

Assertion Coverage:
    Assertions                      17        17         0   100.00%
--------------------------------------------------------------------------------
Name                 File(Line)                   Failure      Pass
                                                  Count        Count
--------------------------------------------------------------------------------
/top/DUTf/sva1/EMPTY_assertion
                     FIFO_SVA.sv(6)                     0          1
/top/DUTf/sva1/ALMOSTEMPTY_assertion
                     FIFO_SVA.sv(10)                    0          1
/top/DUTf/sva1/ALMOSTFULL_assertion
                     FIFO_SVA.sv(14)                    0          1
/top/DUTf/sva1/FULL_assertion
                     FIFO_SVA.sv(18)                    0          1
/top/DUTf/sva1/OVERFLOW_assertion
                     FIFO_SVA.sv(80)                    0          1
/top/DUTf/sva1/UNDERFLOW_assertion
                     FIFO_SVA.sv(81)                    0          1
/top/DUTf/sva1/WR_ACK_HIGH_assertion
                     FIFO_SVA.sv(82)                    0          1
/top/DUTf/sva1/WR_ACK_LOW_assertion
                     FIFO_SVA.sv(83)                    0          1
/top/DUTf/sva1/COUNTER_0_assertion
                     FIFO_SVA.sv(84)                    0          1
/top/DUTf/sva1/COUNTER_INC_10_assertion
                     FIFO_SVA.sv(85)                    0          1
/top/DUTf/sva1/COUNTER_INC_01_assertion
                     FIFO_SVA.sv(86)                    0          1
/top/DUTf/sva1/COUNTER_INC_11_WR_assertion
                     FIFO_SVA.sv(87)                    0          1
/top/DUTf/sva1/COUNTER_INC_11_RD_assertion
                     FIFO_SVA.sv(88)                    0          1
/top/DUTf/sva1/COUNTER_LAT_assertion
                     FIFO_SVA.sv(89)                    0          1
/top/DUTf/sva1/PTR_RST_assertion
                     FIFO_SVA.sv(90)                    0          1
/top/DUTf/sva1/RD_PTR_assertion
                     FIFO_SVA.sv(91)                    0          1
/top/DUTf/sva1/WR_PTR_assertion
                     FIFO_SVA.sv(92)                    0          1
```

Functional Coverage report

```
Directive Coverage:
    Directives                      17        17         0   100.00%

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
-
Name                                    Design Design   Lang File(Line)      Hits Status
                                        Unit   UnitType
--------------------------------------------------------------------------------------------
-
/top/DUTf/sva1/EMPTY_cover              SVA    Verilog  SVA  FIFO_SVA.sv(7)   206 Covered
/top/DUTf/sva1/ALMOSTEMPTY_cover        SVA    Verilog  SVA  FIFO_SVA.sv(11)  229 Covered
/top/DUTf/sva1/ALMOSTFULL_cover         SVA    Verilog  SVA  FIFO_SVA.sv(15)  112 Covered
/top/DUTf/sva1/FULL_cover               SVA    Verilog  SVA  FIFO_SVA.sv(19)   82 Covered
/top/DUTf/sva1/OVERFLOW_cover           SVA    Verilog  SVA  FIFO_SVA.sv(95)  135 Covered
/top/DUTf/sva1/UNDERFLOW_cover          SVA    Verilog  SVA  FIFO_SVA.sv(96)   83 Covered
/top/DUTf/sva1/WR_ACK_HIGH_cover        SVA    Verilog  SVA  FIFO_SVA.sv(97)  994 Covered
/top/DUTf/sva1/WR_ACK_LOW_cover         SVA    Verilog  SVA  FIFO_SVA.sv(98)  135 Covered
/top/DUTf/sva1/COUNTER_0_cover          SVA    Verilog  SVA  FIFO_SVA.sv(99)  208 Covered
/top/DUTf/sva1/COUNTER_INC_10_cover     SVA    Verilog  SVA  FIFO_SVA.sv(100) 705 Covered
/top/DUTf/sva1/COUNTER_INC_01_cover     SVA    Verilog  SVA  FIFO_SVA.sv(101) 113 Covered
/top/DUTf/sva1/COUNTER_INC_11_WR_cover  SVA    Verilog  SVA  FIFO_SVA.sv(102)  55 Covered
/top/DUTf/sva1/COUNTER_INC_11_RD_cover  SVA    Verilog  SVA  FIFO_SVA.sv(103)  37 Covered
/top/DUTf/sva1/COUNTER_LAT_cover        SVA    Verilog  SVA  FIFO_SVA.sv(104) 126 Covered
/top/DUTf/sva1/PTR_RST_cover            SVA    Verilog  SVA  FIFO_SVA.sv(105) 208 Covered
/top/DUTf/sva1/RD_PTR_cover             SVA    Verilog  SVA  FIFO_SVA.sv(106) 384 Covered
/top/DUTf/sva1/WR_PTR_cover             SVA    Verilog  SVA  FIFO_SVA.sv(107) 994 Covered
```

The excluded cross-cover bins are due to impossible cases (wr_en = 0 and wr_ack = 1)

Code Coverage report

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         20        20         0   100.00%

================================Toggle Details================================

Toggle Coverage for instance /\top#DUTf  --

                                          Node      1H->0L      0L->1H  "Coverage"
                                          ----------------------------------------
                                    count[3-0]           1           1      100.00
                                   rd_ptr[2-0]           1           1      100.00
                                   wr_ptr[2-0]           1           1      100.00

Total Node Count     =         10
Toggled Node Count   =         10
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (20 of 20 bins)


DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Hits Status
                                        Unit   UnitType
--------------------------------------------------------------------------------------------
/\top#DUTf /sva1/EMPTY_cover            SVA    Verilog  SVA  FIFO_SVA.sv(7)   206 Covered
/\top#DUTf /sva1/ALMOSTEMPTY_cover      SVA    Verilog  SVA  FIFO_SVA.sv(11)  229 Covered
/\top#DUTf /sva1/ALMOSTFULL_cover       SVA    Verilog  SVA  FIFO_SVA.sv(15)  112 Covered
/\top#DUTf /sva1/FULL_cover             SVA    Verilog  SVA  FIFO_SVA.sv(19)   82 Covered
/\top#DUTf /sva1/OVERFLOW_cover         SVA    Verilog  SVA  FIFO_SVA.sv(95)  135 Covered
/\top#DUTf /sva1/UNDERFLOW_cover        SVA    Verilog  SVA  FIFO_SVA.sv(96)   83 Covered
/\top#DUTf /sva1/WR_ACK_HIGH_cover      SVA    Verilog  SVA  FIFO_SVA.sv(97)  994 Covered
/\top#DUTf /sva1/WR_ACK_LOW_cover       SVA    Verilog  SVA  FIFO_SVA.sv(98)  135 Covered
/\top#DUTf /sva1/COUNTER_0_cover        SVA    Verilog  SVA  FIFO_SVA.sv(99)  208 Covered
/\top#DUTf /sva1/COUNTER_INC_10_cover   SVA    Verilog  SVA  FIFO_SVA.sv(100) 705 Covered
/\top#DUTf /sva1/COUNTER_INC_01_cover   SVA    Verilog  SVA  FIFO_SVA.sv(101) 113 Covered
/\top#DUTf /sva1/COUNTER_INC_11_WR_cover SVA   Verilog  SVA  FIFO_SVA.sv(102)  55 Covered
/\top#DUTf /sva1/COUNTER_INC_11_RD_cover SVA   Verilog  SVA  FIFO_SVA.sv(103)  37 Covered
/\top#DUTf /sva1/COUNTER_LAT_cover      SVA    Verilog  SVA  FIFO_SVA.sv(104) 126 Covered
/\top#DUTf /sva1/PTR_RST_cover          SVA    Verilog  SVA  FIFO_SVA.sv(105) 208 Covered
/\top#DUTf /sva1/RD_PTR_cover           SVA    Verilog  SVA  FIFO_SVA.sv(106) 384 Covered
/\top#DUTf /sva1/WR_PTR_cover           SVA    Verilog  SVA  FIFO_SVA.sv(107) 994 Covered
```

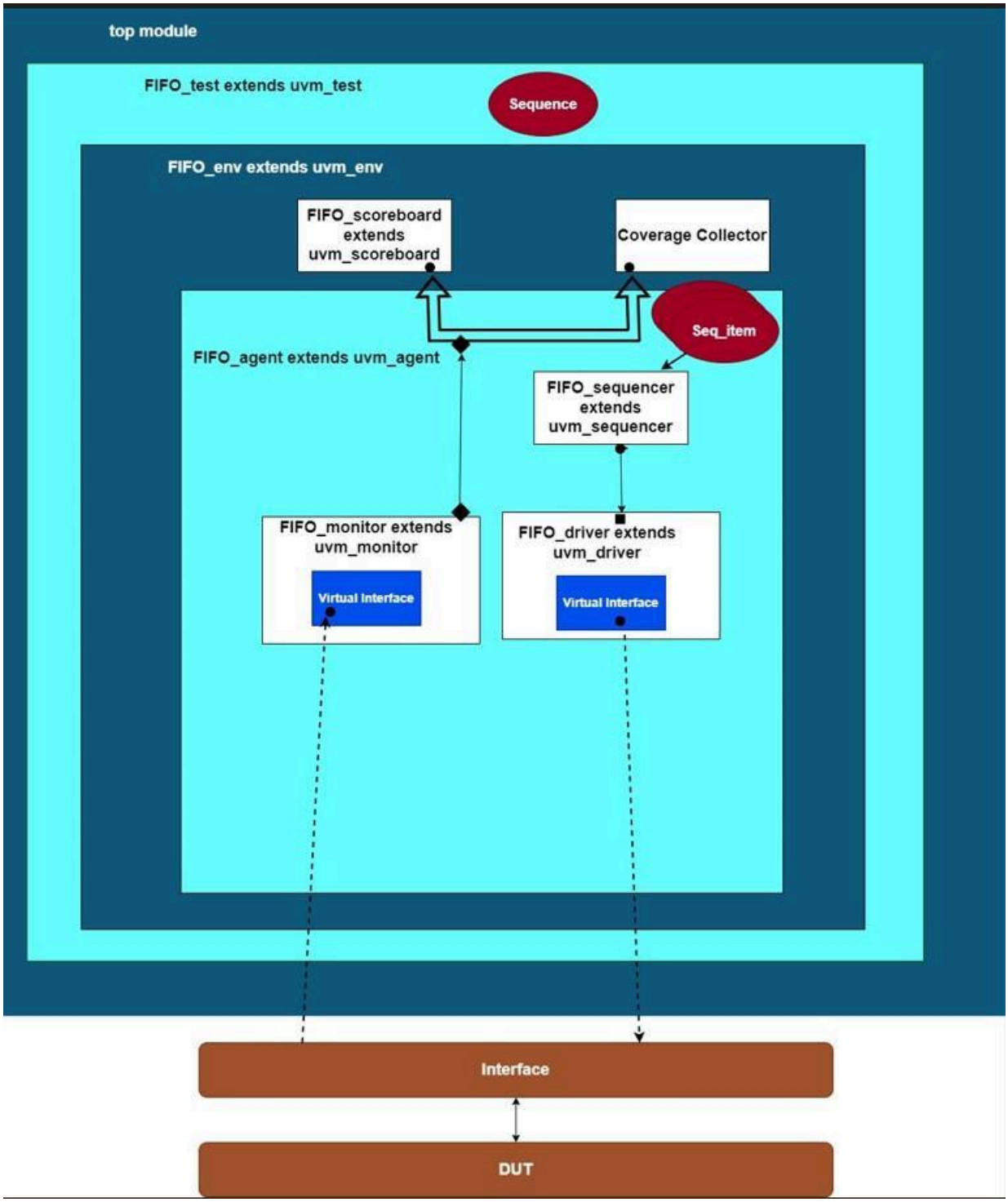
```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      27        27         0   100.00%

================================Statement Details================================

Statement Coverage for instance /\top#DUTf  --

    Line         Item                     Count     Source
    ----         ----                     -----     ------
  File FIFO.sv
    1                                               module FIFO(fifo_if.DUT fifoif);
    2
    3                                               parameter FIFO_DEPTH = 8;
    4                                               localparam max_fifo_addr = $clog2(FIFO_DEPTH);
    5
    6                                               reg [fifoif.FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];
    7
    8                                               reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
    9                                               reg [max_fifo_addr:0] count;
    10
    11              1                        2187     always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
    12                                                      if (!fifoif.rst_n) begin    //resetting all values to 0
    13              1                         393               wr_ptr <= 0;
    14              1                         393               fifoif.wr_ack <= 0;
    15              1                         393               fifoif.overflow <= 0;
    16                                                      end else if (fifoif.wr_en && count < FIFO_DEPTH) begin   //modified the condition to check if the FIFO is fifoif.full or not
    17              1                        1108               mem[wr_ptr] <= fifoif.data_in;
    18              1                        1108               fifoif.wr_ack <= 1;
    19              1                        1108               wr_ptr <= wr_ptr + 1;
    20                                                      end else if (fifoif.wr_en && count == FIFO_DEPTH) begin
    21              1                         155               fifoif.wr_ack <= 0;   // Acknowledge that write wasn't successful
    22              1                         155               fifoif.overflow <= 1; // Set fifoif.overflow if trying to write when fifoif.full
    23                                                      end else begin
    24              1                         531               fifoif.wr_ack <= 0;
    25              1                         531               fifoif.overflow <= 0;
    26                                                      end
    27                                                  end
    28
```

```
Directive Coverage:
    Directives                      17        17         0   100.00%

DIRECTIVE COVERAGE:
-------------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Hits Status
                                        Unit   UnitType
-------------------------------------------------------------------------------------------------
/\top#DUTf /sva1/EMPTY_cover            SVA    Verilog  SVA  FIFO_SVA.sv(7)   206 Covered
/\top#DUTf /sva1/ALMOSTEMPTY_cover      SVA    Verilog  SVA  FIFO_SVA.sv(11)  229 Covered
/\top#DUTf /sva1/ALMOSTFULL_cover       SVA    Verilog  SVA  FIFO_SVA.sv(15)  112 Covered
/\top#DUTf /sva1/FULL_cover             SVA    Verilog  SVA  FIFO_SVA.sv(19)   82 Covered
/\top#DUTf /sva1/OVERFLOW_cover         SVA    Verilog  SVA  FIFO_SVA.sv(95)  135 Covered
/\top#DUTf /sva1/UNDERFLOW_cover        SVA    Verilog  SVA  FIFO_SVA.sv(96)   83 Covered
/\top#DUTf /sva1/WR_ACK_HIGH_cover      SVA    Verilog  SVA  FIFO_SVA.sv(97)  994 Covered
/\top#DUTf /sva1/WR_ACK_LOW_cover       SVA    Verilog  SVA  FIFO_SVA.sv(98)  135 Covered
/\top#DUTf /sva1/COUNTER_0_cover        SVA    Verilog  SVA  FIFO_SVA.sv(99)  208 Covered
/\top#DUTf /sva1/COUNTER_INC_10_cover   SVA    Verilog  SVA  FIFO_SVA.sv(100) 705 Covered
/\top#DUTf /sva1/COUNTER_INC_01_cover   SVA    Verilog  SVA  FIFO_SVA.sv(101) 113 Covered
/\top#DUTf /sva1/COUNTER_INC_11_WR_cover SVA   Verilog  SVA  FIFO_SVA.sv(102)  55 Covered
/\top#DUTf /sva1/COUNTER_INC_11_RD_cover SVA   Verilog  SVA  FIFO_SVA.sv(103)  37 Covered
/\top#DUTf /sva1/COUNTER_LAT_cover      SVA    Verilog  SVA  FIFO_SVA.sv(104) 126 Covered
/\top#DUTf /sva1/PTR_RST_cover          SVA    Verilog  SVA  FIFO_SVA.sv(105) 208 Covered
/\top#DUTf /sva1/RD_PTR_cover           SVA    Verilog  SVA  FIFO_SVA.sv(106) 384 Covered
/\top#DUTf /sva1/WR_PTR_cover           SVA    Verilog  SVA  FIFO_SVA.sv(107) 994 Covered
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       1         1         0   100.00%
```

```
Condition Coverage:
    Enabled Coverage              Bins    Covered    Misses  Coverage
    ----------------              ----       ----    ------  --------
    Conditions                       4          4         0   100.00%

================================Condition Details================================

Condition Coverage for instance /\top#DUTf /sva1 --

  File FIFO_SVA.sv
----------------Focused Condition View-------------------
Line       5 Item    1  (FIFO.count == 0)
Condition totals: 1 of 1 input term covered = 100.00%

         Input Term   Covered  Reason for no coverage   Hint
        -----------  --------  ------------------------  --------------
  (FIFO.count == 0)         Y

     Rows:       Hits  FEC Target            Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  (FIFO.count == 0)_0   -
  Row   2:          1  (FIFO.count == 0)_1   -

----------------Focused Condition View-------------------
Line       9 Item    1  (FIFO.count == 1)
Condition totals: 1 of 1 input term covered = 100.00%
```

```
Branch Coverage:
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Branches                       23        23         0   100.00%

================================Branch Details================================

Branch Coverage for instance /\top#DUTf

    Line         Item                     Count     Source
    ----         ----                     -----     ------
  File FIFO.sv
------------------------------------IF Branch------------------------------------
    12                                    2187     Count coming in to IF
    12              1                      393            if (!fifoif.rst_n) begin    //resetting all values to 0
    16              1                     1108            end else if (fifoif.wr_en && count < FIFO_DEPTH) begin   //modified the condition to check if the FIFO is fifoif.full or not
    20              1                      155            end else if (fifoif.wr_en && count == FIFO_DEPTH) begin
    23              1                      531            end else begin
Branch totals: 4 hits of 4 branches = 100.00%

------------------------------------IF Branch------------------------------------
    30                                    2187     Count coming in to IF
    30              1                      393        if (!fifoif.rst_n) begin
    34              1                      423       else if (fifoif.rd_en && count != 0) begin
    38              1                       95         else if (fifoif.rd_en && count == 0) begin
                                          1276     All False Count
Branch totals: 4 hits of 4 branches = 100.00%

------------------------------------IF Branch------------------------------------
    44                                    2001     Count coming in to IF
    44              1                      384        if (!fifoif.rst_n) begin
    47              1                     1617        else begin
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    48                                    1617     Count coming in to IF
    48              1                      787              if      ( ({fifoif.wr_en, fifoif.rd_en} == 2'b10) && count < FIFO_DEPTH)
    50              1                      125              else if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b01) && count != 0)
    52              1                       42               else if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b11) && count == FIFO_DEPTH)  //read only
    54              1                       65               else if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b11) && count == 0)  //write only
                                          598     All False Count
Branch totals: 5 hits of 5 branches = 100.00%
```

How the UVM testbench work



The UVM testbench for the FIFO module operates through several stages, starting from the top module and moving through driving the interface, monitoring, and analyzing the output.

Top-Level Module

The test starts with the `FIFO_test` class, which extends `uvm_test`. This is where I handle the configuration, environment setup, and all the sequences that drive the FIFO. In the `build_phase`, I create instances of:

- `FIFO_env`, which contains the environment components like the agent,
- `fifo_config`, which holds the testbench configuration, including the virtual interface,
- Various sequences (`seq_reset`, `wr_seq`, `rd_seq`, `rw_seq`) that define specific operations to be applied to the FIFO.

I ensure that the virtual interface `FIFO_if` is correctly retrieved from the configuration database using `uvm_config_db`. This interface is essential for connecting the testbench to the DUT.

Driving the Interface

Once the environment is set up, the next step is driving the interface through the agent. The agent, which is created in `FIFO_env`, consists of:

- **Driver:** This component takes transactions from the sequencer and converts them into low-level signals on the FIFO interface.
- **Sequencer:** I control the execution of different sequences here. The sequencer feeds transactions to the driver based on the current sequence (reset, write-only, read-only, or read-write). These sequences are started in the `run_phase`, where I raise objections to keep the simulation running until all sequences are done.

Monitoring the Output

The monitor, also part of the agent, passively observes the interface signals. It captures any activity happening on the interface, like data being written or read, and packages this information into transactions. This way, I can monitor the behavior of the FIFO without influencing the signals directly.

Analyzing the Output (Scoreboard)

Finally, the scoreboard takes the monitored transactions and compares the actual results with the expected behavior. This is where I verify that the FIFO is functioning correctly, ensuring that the data written into the FIFO matches the data being read out. If there's any mismatch or if specific properties are violated (like FIFO overflow or underflow), the scoreboard will flag those errors.

I provide feedback during the test through UVM messages (`uvm_info`), which helps in tracking progress and diagnosing any issues. Once all the sequences are complete, I drop the objection to signal the end of the simulation.

This testbench structure allows me to thoroughly verify the FIFO module, ensuring that it behaves as expected under various conditions.

Verification Plan:

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	When the reset is asserted, the output signals are all desserted except the empty flag	Directed at the start of the simulation, then randomized with constraint that drive the reset to be off most of the simulation time		A checker through a reference model task to make sure the output is correct, with assertion: PTR_RST_assertion: Ensures that both the read and write pointers are reset to 0 after a reset.
FIFO_2	When the wr_en signal is asserted, rd_en is desserted and data_in take randomized value while the memory isn't full: the memory with address = wr_ptr will take the data_in value, wr_ack will be asserted and counter is incremented	Randomization Under wr_only constraint that the value of wr_en to be on, rd_en to be off and reset signal to be desserted	Coverpoint for wr_en, rd_en and wr_ack	A checker through a reference model task and assertions for output flags to make sure the output is correct, WR_PTR_assertion: Ensures that the write pointer increments correctly when a write operation is performed and the FIFO is not full.
FIFO_3	When the rd_en signal is asserted, wr_en is desserted while the memory isn't empty: the data_out = memory with address = rd_ptr	Randomization Under rd_only constraint that the value of wr_en to be on, rd_en to be off and reset signal to be desserted	Coverpoint for wr_en, rd_en and wr_ack	A checker through a reference model task and assertions for output flags to make sure the output is correct, with assertions: WR_ACK_LOW_assertion: Checks that wr_ack is low when a write operation is attempted on a full FIFO (wr_en = 1, full = 1). RD_PTR_assertion: Ensures that the read pointer increments correctly when a read operation is performed and the FIFO is not empty.
FIFO_4	When the rd_en signal and wr_en are asserted, while the memory isn't empty or full: the data_out = memory with address = rd_ptr and the memory with address = wr_ptr will take the data_in value, wr_ack will be asserted and counter won't be affected	Randomization Under constraints that the wr_en to be on 70% of the time while rd_en is on for 30% of the time	Coverpoint for wr_en, rd_en and wr_ack	A checker through a reference model task and assertions for output flags to make sure the output is correct, with assertion: WR_ACK_HIGH_assertion: Checks that wr_ack is high when a write operation is performed successfully (i.e., wr_en = 1, FIFO is not full).
FIFO_5	When the rd_en signal and wr_en are asserted, while the memory is empty: the memory with address = wr_ptr will take the data_in value, wr_ack will be asserted and counter will be incremented	Randomization Under constraints that the wr_en to be on 70% of the time while rd_en is on for 30% of the time	Coverpoint for wr_en, rd_en and empty and for wr_en, rd_en and almost empty	A checker through a reference model task and assertions for output flags to make sure the output is correct, ALMOSTEMPTY_assertion: Asserts that when there is one element in the FIFO (count == 1), the correct status flags are set (almostempty = 1, empty = 0, almostfull = 0).
FIFO_6	When the rd_en signal and wr_en are asserted, while the memory is full: the data_out = memory with address = rd_ptr and counter will be decremented	Randomization Under constraints that the wr_en to be on 70% of the time while rd_en is on for 30% of the time	Coverpoint for wr_en, rd_en and full and for wr_en, rd_en and almost full	A checker through a reference model task and assertions for output flags to make sure the output is correct,ALMOSTFULL_assertion: Asserts that when the FIFO is almost full (count == FIFO_DEPTH-1), the correct status flags are set (almostfull = 1, full = 0, empty = 0).
FIFO_7	When the wr_en signal is asserted, rd_en is desserted and data_in take randomized value while the memory is full: overflow will be asserted, wr_ack will be desserted and counter isn't changed	Randomization Under constraints that the wr_en to be on 70% of the time while rd_en is on for 30% of the time	Coverpoint for wr_en, rd_en and overflow	A checker through a reference model task and assertions for output flags to make sure the output is correct, OVERFLOW_assertion: Checks that when the FIFO is full (full = 1) and a write enable (wr_en) occurs, the overflow flag is raised.
FIFO_8	When the rd_en signal is asserted, wr_en is desserted while the memory is empty: underflow is asserted and counter isn't changed	Randomization Under constraints that the wr_en to be on 70% of the time while rd_en is on for 30% of the time	Coverpoint for wr_en, rd_en and underflow	A checker through a reference model task and assertions for output flags to make sure the output is correct,UNDERFLOW_assertion: Checks that when the FIFO is empty (empty = 1) and a read enable (rd_en) occurs, the underflow flag is raised.

Bug report:

Bug Number	Bug Description	Original Code	Fixed Code
1	Missing reset behavior for `wr_ack` and	No reset logic for `wr_ack` and	Reset logic added for `wr_ack` and `overflow`

	`overflow`. These signals were not being reset when `rst_n` was low.	`overflow` in the `always @(posedge clk or negedge rst_n)` block.	when `rst_n` is asserted.
2	Incorrect full condition handling during write operations. The overflow was not being flagged when FIFO was full.	No logic to handle the condition when the FIFO is full and a write attempt is made.	Added condition to check if `count == FIFO_DEPTH` and properly set `overflow` and `wr_ack`.
3	Missing underflow detection during read operations on an empty FIFO.	No underflow logic when attempting to read from an empty FIFO.	Added logic to detect underflow when `rd_en` is high, but `count` is 0.
4	Incorrect threshold for `almostfull` signal, which triggered at `count == FIFO_DEPTH-2` .	`assign almostfull = (count == FIFO_DEPTH-2) ? 1 : 0;`	Changed `almostfull` threshold to trigger at `count == FIFO_DEPTH-1`, indicating only one slot left in the FIFO.
5	Incorrect FIFO count update during simultaneous read and write operations.	Simultaneous read/write conditions not handled properly. No clear update logic for `count` when both `wr_en` and `rd_en` are high, especially when FIFO is full or empty.	Added conditions for simultaneous read/write operations. The count is now incremented or decremented based on whether the FIFO is full or empty.

Assertions Table:

Feature	Assertion
FIFO is empty, and status signals reflect empty condition	@(posedge fifoif.clk) (FIFO.count == 0)
FIFO has 1 item, and status signals reflect almost empty condition	@(posedge fifoif.clk) (FIFO.count == 1)
FIFO is almost full, and status signals reflect almost full condition	@(posedge fifoif.clk) (FIFO.count == FIFO_DEPTH-1)
FIFO is full, and status signals reflect full condition	@(posedge fifoif.clk) (FIFO.count == FIFO_DEPTH)
Overflow occurs when writing to a full FIFO	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) (fifoif.full && fifoif.wr_en)
Underflow occurs when reading from an empty FIFO	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) (fifoif.empty && fifoif.rd_en)
Wr_ack high when FIFO is not full and write enable is asserted	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) (fifoif.wr_en && (FIFO.count < FIFO_DEPTH) && !fifoif.full)
Wr_ack low when FIFO is full and write enable is asserted	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) (fifoif.wr_en && fifoif.full)
FIFO count increments by 1 when only writing to a non-full FIFO	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) (({fifoif.wr_en, fifoif.rd_en} == 2'b10) && !fifoif.full)
FIFO count decrements by 1 when only reading from a non-empty FIFO	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) (({fifoif.wr_en, fifoif.rd_en} == 2'b01) && !fifoif.empty)
FIFO count increments by 1 when both reading and writing, and FIFO is empty	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) (({fifoif.wr_en, fifoif.rd_en} == 2'b11) && fifoif.empty)
FIFO count decrements by 1 when	@(posedge fifoif.clk) disable iff

both reading and writing, and FIFO is full	(!fifoif.rst_n) (({fifoif.wr_en, fifoif.rd_en} == 2'b11) && fifoif.full)
FIFO count remains the same when reading from an empty or writing to a full FIFO	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) ((({fifoif.wr_en, fifoif.rd_en} == 2'b01) && fifoif.empty)
Read pointer resets when FIFO is reset	@(posedge fifoif.clk) (!fifoif.rst_n)
Read pointer increments when reading from a non-empty FIFO	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) (fifoif.rd_en && (FIFO.count != 0))
Write pointer increments when writing to a non-full FIFO	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) (fifoif.wr_en && (FIFO.count < FIFO_DEPTH))

Code Snippets:

Fifo_agent:

```
package FIFO_agent_pkg;
import uvm_pkg::*;
import FIFO_config_pkg::*;
import sequencer_fifo_pkg::*;
import FIFO_Monitor_pkg::*;
import FIFO_driver_pkg::*;
import FIFO_seq_item_pkg::*;
`include "uvm_macros.svh"

class FIFO_agent extends uvm_agent;
`uvm_component_utils(FIFO_agent)
  fifo_sequencer seqr;
fifo_driver drv;
  FIFO_Monitor mon;
fifo_config fifo_cfg;
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)::get(this, "", "fifo_cfg",
fifo_cfg)) begin //set fl test
`uvm_fatal("build_phase", "Driver unable to get config") end
seqr = fifo_sequencer::type_id::create("seqr", 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);
drv.fifo_vif = fifo_cfg.fifo_vif;
        mon.fifo_vif = fifo_cfg.fifo_vif;
drv.seq_item_port.connect(seqr.seq_item_export);
        mon.mon_ap.connect(agt_ap);
    endfunction
endclass
endpackage
```

Fifo_config:

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

class fifo_config extends uvm_object;
`uvm_object_utils(fifo_config )
virtual fifo_if fifo_vif;

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

endclass
endpackage
```

Fifo_coverage:

```
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
  cov_seq_item;

  covergroup covCode;
  // Cross coverage between wr_en, rd_en and all control signals
      cross cov_seq_item.wr_en, cov_seq_item.rd_en, cov_seq_item.wr_ack;
      cross cov_seq_item.wr_en, cov_seq_item.rd_en, cov_seq_item.overflow;
      cross cov_seq_item.wr_en, cov_seq_item.rd_en, cov_seq_item.full;
      cross cov_seq_item.wr_en, cov_seq_item.rd_en, cov_seq_item.empty;
      cross cov_seq_item.wr_en, cov_seq_item.rd_en,
cov_seq_item.almostfull;
      cross cov_seq_item.wr_en, cov_seq_item.rd_en,
cov_seq_item.almostempty;
      cross cov_seq_item.wr_en, cov_seq_item.rd_en,
cov_seq_item.underflow;
      endgroup

  function new(string name = "FIFO_Coverage", uvm_component parent =
null);
    super.new(name, parent);
    covCode = new();
  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);
    cov_export.connect(cov_fifo.analysis_export);
```

```
endfunction

  task run_phase(uvm_phase phase);
    super.run_phase(phase);
forever begin
cov_fifo.get(cov_seq_item);
      covCode.sample();
end
  endtask
endclass
endpackage
```

Fifo_driver:

```
package FIFO_driver_pkg;
import uvm_pkg::*;
import FIFO_seq_item_pkg::*;
import FIFO_config_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_config fifo_cfg;
FIFO_seq_item stim_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
stim_seq_item     =     FIFO_seq_item::type_id::create("stim_seq_item");
        seq_item_port.get_next_item(stim_seq_item);
fifo_vif.rst_n     =     stim_seq_item.rst_n;
        fifo_vif.wr_en = stim_seq_item.wr_en;
        fifo_vif.rd_en = stim_seq_item.rd_en;
fifo_vif.data_in     =     stim_seq_item.data_in;
        @(negedge fifo_vif.clk);
```

```
seq_item_port.item_done();
        `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(),
UVM_HIGH)
end
    endtask
endclass
endpackage
```

Fifo_environment:

```
package FIFO_env_pkg;
import uvm_pkg::*;
import FIFO_agent_pkg::*;
import FIFO_Coverage_pkg::*;
import FIFO_Scoreboard_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
```

Fifo_if:

```
interface fifo_if(clk);
parameter FIFO_WIDTH = 16;
input clk;
logic [FIFO_WIDTH-1:0] data_in;
logic rst_n, wr_en, rd_en;
logic [FIFO_WIDTH-1:0] data_out;
logic wr_ack, overflow;
logic full, empty, almostfull, almostempty, underflow;
// bit [3:0] fifo_count;

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

Fifo_monitor:

```
package FIFO_Monitor_pkg;
import uvm_pkg::*;
import FIFO_seq_item_pkg:: *;
import shared_pkg::*;
`include "uvm_macros.svh"

class FIFO_Monitor extends uvm_monitor;
`uvm_component_utils(FIFO_Monitor)
    virtual fifo_if fifo_vif;
FIFO_seq_item rsp_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
rsp_seq_item = FIFO_seq_item::type_id::create("rsp_seq_item");

@(negedge fifo_vif.clk);
rsp_seq_item.data_in = fifo_vif.data_in; rsp_seq_item.rst_n =
            fifo_vif.rst_n;
rsp_seq_item.wr_en = fifo_vif.wr_en; rsp_seq_item.rd_en =
            fifo_vif.rd_en;
rsp_seq_item.data_out = fifo_vif.data_out; rsp_seq_item.wr_ack =
            fifo_vif.wr_ack;
rsp_seq_item.overflow = fifo_vif.overflow; rsp_seq_item.full =
            fifo_vif.full;
rsp_seq_item.empty = fifo_vif.empty;
rsp_seq_item.almostfull = fifo_vif.almostfull;
rsp_seq_item.almostempty = fifo_vif.almostempty; rsp_seq_item.underflow =
            fifo_vif.underflow;

mon_ap.write(rsp_seq_item);
`uvm_info("run_phase", rsp_seq_item.convert2string(),
UVM_HIGH)
        end
    endtask
endclass
endpackage
```

Fifo_scoreboard:

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

    bit [15:0] mem_queue [$];
    bit [15:0] data_out_ref;
    bit full_ref, empty_ref, almostfull_ref, almostempty_ref,
underflow_ref,wr_ack_ref,overflow_ref;

    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 run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            sb_fifo.get(seq_item_sb);
            reference_model(seq_item_sb);

            if(seq_item_sb.data_out != data_out_ref) begin
                `uvm_error("run_phase", $sformatf("Comparison Fail: DUT:
%s while  Data_out_ref: %0d", seq_item_sb.convert2string(),
data_out_ref));
```

```
                error_count++;
            end
            else begin
                `uvm_info("run_phase", $sformatf("Comparison Pass: DUT: %s
", seq_item_sb.convert2string()), UVM_HIGH);
                correct_count++;
            end
        end
    endtask

    task reference_model(FIFO_seq_item F_txn);
        if (F_txn.rst_n == 0) begin
            // data_out_ref = 16'b0;
            full_ref = 0;
            empty_ref = 1;
            almostfull_ref = 0;
            almostempty_ref = 0;
            underflow_ref = 0;
            wr_ack_ref = 0;
            overflow_ref = 0;
            mem_queue.delete();
        end
        else begin
            // *****************Write and
Read*****************************
            if ({F_txn.rd_en , F_txn.wr_en} == 2'b11)begin
                if ($size(mem_queue) == 0)begin //empty
                    mem_queue.push_back(F_txn.data_in);
                    wr_ack_ref = 1 ;
                    overflow_ref = 0;
                    underflow_ref = 1;
                end
                else if ($size(mem_queue) == 8) begin //full
                    data_out_ref = mem_queue.pop_front();
                    underflow_ref = 0;
                    overflow_ref = 1;
                    wr_ack_ref = 0 ;
                end
                else begin
                    mem_queue.push_back(F_txn.data_in);
```

```
                                                    data_out_r
                                                    ef =
                                                    mem_queue.
                                                    pop_front(
                    end                             );
                end                                 wr_ack_ref
                                                    = 1 ;
                                                    overflow_ref = 0;
                                                    underflow_ref = 0;
    else begin
        {overflow_ref , wr_ack_ref , underflow_ref } = 3'b0 ;
         if (F_txn.wr_en) begin
    // *****************Write only*****************************
            if ($size(mem_queue) < 8) begin
                mem_queue.push_back(F_txn.data_in)
                ; wr_ack_ref = 1 ;
                overflow_ref = 0;
            end
            else begin
                wr_ack_ref = 0 ;
                overflow_ref = 1;
            end
        end
    // *****************Read only*****************************
        else if (F_txn.rd_en)begin
            if ($size(mem_queue) > 0) begin
                data_out_ref = mem_queue.pop_front();
                underflow_ref = 0;
            end
            else begin
                underflow_ref = 1;
            end
        end

    end

end

full_ref = ($size(mem_queue) == 8)? 1 : 0 ;
empty_ref = ($size(mem_queue) == 0)? 1 : 0 ;
almostfull_ref = ($size(mem_queue) == 7)? 1 : 0 ;
almostempty_ref = ($size(mem_queue) == 1)? 1 : 0 ;
```

```
    endtask
    // function void print();
    //     $display("data_out_ref =%0d ,wr_ack_ref =%0d , overflow_ref
=%0d , full_ref =%0d , empty_ref =%0d , almostfull_Ref =%0d ,
almostempty_ref =%0d , underflow_ref =%0d     ",data_out_ref
,wr_ack_ref , overflow_ref ,full_ref     ,empty_ref , almostfull_ref
,almostempty_ref , underflow_ref);
    // endfunction

    function void report_phase(uvm_phase phase);
    super.report_phase(phase);
    `uvm_info("report_phase", $sformatf("Total correct:%0d
",correct_count),UVM_MEDIUM);
    `uvm_info("report_phase", $sformatf("Total error:%0d
",error_count),UVM_MEDIUM);
    endfunction

endclass
endpackage
```

Fifo_seq_item:

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

class FIFO_seq_item extends uvm_sequence_item;
`uvm_object_utils(FIFO_seq_item)
bit clk;
parameter FIFO_WIDTH = 16;
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;


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

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

function string convert2string_stimulus();
return $sformatf("FIFO_seq_item: data_in=%0h, rst_n=%0d, wr_en=%0d,
rd_en=%0d", data_in, rst_n, wr_en, rd_en);
endfunction

    constraint reset_signal { rst_n dist {0:=10, 1:=90};
    } constraint wr_signal { wr_en dist {1:=70, 0:=30}; }
    constraint rd_signal  { rd_en dist {1:=30, 0:=70}; }

    // constraint wr_only{ wr_en == 1; rst_n==1; rd_en == 0; }
    // constraint rd_only{ rd_en == 1; rst_n==1; wr_en == 0; }
endclass
endpackage
```

Fifo_sequence:

```
package sequence_fifo_pkg;
import uvm_pkg::*;
import shared_pkg::*;
import FIFO_seq_item_pkg::*;
`include "uvm_macros.svh"

class FIFO_sequence_reset extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(FIFO_sequence_reset)
FIFO_seq_item seq_item;
  function new(string name = "FIFO_sequence_reset");
    super.new(name);
endfunction

task body();
repeat(50) begin
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 = 16'h0;

    finish_item(seq_item);
    end
    endtask
endclass

class write_only_sequence extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(write_only_sequence)
FIFO_seq_item seq_item;
  function new(string name = "write_only_sequence");
    super.new(name);
    endfunction

    task body();
    repeat(1000) begin
    seq_item = FIFO_seq_item::type_id::create("seq_item");
    start_item(seq_item);

    assert(seq_item.randomize());

    seq_item.rst_n = 1;
    seq_item.rd_en = 0;
    seq_item.wr_en = 1;

    finish_item(seq_item);
    end
    endtask
endclass

class read_only_sequence extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(read_only_sequence)
FIFO_seq_item seq_item;
  function new(string name = "read_only_sequence");
    super.new(name);
    endfunction
```

```
    task body();
    repeat(500) begin
    seq_item = FIFO_seq_item::type_id::create("seq_item");
    start_item(seq_item);

    assert(seq_item.randomize());
    seq_item.rst_n = 1;
    seq_item.rd_en = 1;
    seq_item.wr_en = 0;

    finish_item(seq_item);
    end
    endtask
endclass

class read_write_sequence extends uvm_sequence #(FIFO_seq_item);
`uvm_object_utils(read_write_sequence)
FIFO_seq_item seq_item;
  function new(string name = "read_write_sequence");
    super.new(name);
    endfunction

    task body();
    repeat(500) begin
    seq_item = FIFO_seq_item::type_id::create("seq_item");
    start_item(seq_item);

    assert(seq_item.randomize());
    seq_item.rst_n = 1;
    seq_item.rd_en = 1;
    seq_item.wr_en = 1;

    finish_item(seq_item);
    end
    endtask
endclass

endpackage
```

Fifo_sequencer:

```
package sequencer_fifo_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
```

Fifo_assertions:

```
module SVA(fifo_if.DUT fifoif);
    parameter FIFO_DEPTH = 8;
// Assertions for Combinational Outputs
    always_comb begin
if (FIFO.count == 0) begin
EMPTY_assertion : assert (fifoif.empty && !fifoif.full &&
!fifoif.almostempty && !fifoif.almostfull) else $display("EMPTY_assertion
fail");
EMPTY_cover : cover    (fifoif.empty && !fifoif.full &&
!fifoif.almostempty && !fifoif.almostfull)    ;
        end
if (FIFO.count == 1) begin
            ALMOSTEMPTY_assertion : assert (!fifoif.empty && !fifoif.full
&& fifoif.almostempty && !fifoif.almostfull) else
$display("ALMOSTFULL_assertion fail");
            ALMOSTEMPTY_cover      : cover     (!fifoif.empty &&
!fifoif.full && fifoif.almostempty && !fifoif.almostfull) ;
end
if (FIFO.count == FIFO_DEPTH-1) begin
```

```
            ALMOSTFULL_assertion : assert (!fifoif.empty && !fifoif.full
&& !fifoif.almostempty && fifoif.almostfull) else
$display("ALMOSTFULL_assertion fail");
            ALMOSTFULL_cover     : cover  (!fifoif.empty &&
!fifoif.full && !fifoif.almostempty && fifoif.almostfull);
        end
        if (FIFO.count == FIFO_DEPTH) begin
            FULL_assertion : assert (!fifoif.empty && fifoif.full &&
!fifoif.almostempty && !fifoif.almostfull) else $display("FULL_assertion
fail");
            FULL_cover     : cover  (!fifoif.empty && fifoif.full &&
!fifoif.almostempty && !fifoif.almostfull);
        end
    end

    // Assertions for Overflow and Underflow
    property OVERFLOW_FIFO;
        @(posedge fifoif.clk) disable iff (!fifoif.rst_n) (fifoif.full &
fifoif.wr_en) |=> (fifoif.overflow);
    endproperty

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

    // Assertions for fifoif.wr_ack
    property WR_ACK_HIGH;
        @(posedge fifoif.clk) disable iff (!fifoif.rst_n) (fifoif.wr_en &&
(FIFO.count < FIFO_DEPTH) && !fifoif.full) |=> (fifoif.wr_ack);
    endproperty

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

    // Assertions for The Counter
    property COUNT_0;
        @(posedge fifoif.clk) (!fifoif.rst_n) |=> (FIFO.count == 0);
```

```
    endproperty

    property COUNT_INC_10;
        @(posedge fifoif.clk) disable iff (!fifoif.rst_n) (({fifoif.wr_en,
fifoif.rd_en} == 2'b10) && !fifoif.full) |=> (FIFO.count ==
$past(FIFO.count) + 1);
    endproperty

    property COUNT_INC_01;
        @(posedge fifoif.clk) disable iff (!fifoif.rst_n) (({fifoif.wr_en,
fifoif.rd_en} == 2'b01) && !fifoif.empty) |=> (FIFO.count ==
$past(FIFO.count) - 1);
    endproperty

    property COUNT_INC_11_WR;
        @(posedge fifoif.clk) disable iff (!fifoif.rst_n) (({fifoif.wr_en,
fifoif.rd_en} == 2'b11) && fifoif.empty) |=> (FIFO.count ==
$past(FIFO.count) + 1);
    endproperty

    property COUNT_INC_11_RD;
        @(posedge fifoif.clk) disable iff (!fifoif.rst_n) (({fifoif.wr_en,
fifoif.rd_en} == 2'b11) && fifoif.full) |=> (FIFO.count ==
$past(FIFO.count) - 1);
    endproperty

    property COUNT_LAT;
        @(posedge fifoif.clk) disable iff (!fifoif.rst_n)
((({fifoif.wr_en, fifoif.rd_en} == 2'b01) && fifoif.empty) ||
(({fifoif.wr_en, fifoif.rd_en} == 2'b10) && fifoif.full)) |=> (FIFO.count
== $past(FIFO.count));
    endproperty

    // Assertions for
    Pointers property
    PTR_RST;
        @(posedge fifoif.clk) (!fifoif.rst_n) |=> (~FIFO.rd_ptr &&
~FIFO.wr_ptr);
    endproperty

    property RD_PTR;
```

```
        @(posedge fifoif.clk) disable iff (!fifoif.rst_n) (fifoif.rd_en &&
(FIFO.count != 0)) |=> (FIFO.rd_ptr == ($past(FIFO.rd_ptr) + 1) %
FIFO_DEPTH);
    endproperty

    property WR_PTR;
        @(posedge fifoif.clk) disable iff (!fifoif.rst_n) (fifoif.wr_en &&
(FIFO.count < FIFO_DEPTH)) |=> (FIFO.wr_ptr == ($past(FIFO.wr_ptr) + 1) %
FIFO_DEPTH);
    endproperty

    // Assert Properties
    OVERFLOW_assertion          : assert property (OVERFLOW_FIFO)    else
$display("OVERFLOW_assertion");
    UNDERFLOW_assertion         : assert property (UNDERFLOW_FIFO)   else
$display("UNDERFLOW_assertion");
    WR_ACK_HIGH_assertion       : assert property (WR_ACK_HIGH)      else
$display("WR_ACK_HIGH_assertion");
    WR_ACK_LOW_assertion        : assert property (WR_ACK_LOW)       else
$display("WR_ACK_LOW_assertion");
    COUNTER_0_assertion         : assert property (COUNT_0)          else
$display("COUNTER_0_assertion");
    COUNTER_INC_10_assertion    : assert property (COUNT_INC_10)     else
$display("COUNTER_INC_WR_assertion fail");
    COUNTER_INC_01_assertion    : assert property (COUNT_INC_01)     else
$display("COUNTER_INC_WR_assertion fail");
    COUNTER_INC_11_WR_assertion : assert property (COUNT_INC_11_WR)  else
$display("COUNTER_INC_WR_assertion fail");
    COUNTER_INC_11_RD_assertion : assert property (COUNT_INC_11_RD)  else
$display("COUNTER_INC_WR_assertion fail");
    COUNTER_LAT_assertion       : assert property (COUNT_LAT)        else
$display("COUNTER_LAT_assertion fail");
    PTR_RST_assertion           : assert property (PTR_RST)          else
$display("PTR_RST_asssertion fail");
    RD_PTR_assertion            : assert property (RD_PTR)           else
$display("RD_PTR_asssertion fail");
    WR_PTR_assertion            : assert property (WR_PTR)           else
$display("WR_PTR_asssertion fail");

    // Cover Properties
```

```
    OVERFLOW_cover           : cover property (OVERFLOW_FIFO);
    UNDERFLOW_cover          : cover property (UNDERFLOW_FIFO);
    WR_ACK_HIGH_cover        : cover property (WR_ACK_HIGH);
    WR_ACK_LOW_cover         : cover property (WR_ACK_LOW);
    COUNTER_0_cover          : cover property (COUNT_0);
    COUNTER_INC_10_cover     : cover property (COUNT_INC_10);
    COUNTER_INC_01_cover     : cover property (COUNT_INC_01);
    COUNTER_INC_11_WR_cover  : cover property (COUNT_INC_11_WR);
    COUNTER_INC_11_RD_cover  : cover property (COUNT_INC_11_RD);
    COUNTER_LAT_cover: cover property (COUNT_LAT);
    PTR_RST_cover     : cover property (PTR_RST);
    RD_PTR_cover      : cover property (RD_PTR);
    WR_PTR_cover      : cover property (WR_PTR);

endmodule
```

Fifo_test:

```
package FIFO_test_pkg;
import uvm_pkg::*;
import FIFO_env_pkg::*;
import FIFO_config_pkg::*;
import sequence_fifo_pkg::*;
`include "uvm_macros.svh"

class FIFO_test extends uvm_test;
`uvm_component_utils(FIFO_test)
  FIFO_env env;
fifo_config fifo_cfg;
  FIFO_sequence_reset seq_reset;
    write_only_sequence wr_seq;
    read_only_sequence rd_seq;
read_write_sequence rw_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::type_id::create("fifo_cfg", this);
        seq_reset = FIFO_sequence_reset::type_id::create("seq_reset");
        wr_seq = write_only_sequence::type_id::create("wr_seq");
        rd_seq = read_only_sequence::type_id::create("rd_seq");
        rw_seq = read_write_sequence::type_id::create("rw_seq");

        if(!uvm_config_db #(virtual fifo_if)::get(this, "", "FIFO_if",
fifo_cfg.fifo_vif))  //set fl top
            `uvm_fatal("build_phase", "Driver unable to get virtual
interface");
           uvm_config_db #(fifo_config)::set(this, "*", "fifo_cfg",
fifo_cfg);

    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        phase.raise_objection(this);

        `uvm_info("run_phase", "Starting FIFO reset sequence", UVM_LOW);
        seq_reset.start(env.agt.seqr);
        `uvm_info("run_phase", "Ending FIFO reset sequence", UVM_LOW);
        `uvm_info("run_phase", "Starting write only sequence", UVM_LOW);
        wr_seq.start(env.agt.seqr);
        `uvm_info("run_phase", "Ending write only sequence", UVM_LOW);
        `uvm_info("run_phase", "Starting read only sequence", UVM_LOW);
        rd_seq.start(env.agt.seqr);
        `uvm_info("run_phase", "Ending read only sequence", UVM_LOW);
        `uvm_info("run_phase", "Starting read write sequence", UVM_LOW);
        rw_seq.start(env.agt.seqr);
        `uvm_info("run_phase", "Ending read write sequence", UVM_LOW);

        phase.drop_objection(this);
    endtask: run_phase
endclass
endpackage
```

Fifo_top:

```
import uvm_pkg::*;
import FIFO_env_pkg::*;
```

```
import FIFO_test_pkg::*;
`include "uvm_macros.svh"

module top();
bit clk;
//clock generation
  initial begin
clk = 0;
    forever
#1 clk = ~clk;
end

fifo_if fifoif(clk);
FIFO DUTf(fifoif);
bind FIFO SVA sva1(fifoif);

initial begin
uvm_config_db #(virtual
fifo_if)::set(null,"uvm_test_top","FIFO_if",fifoif);
    run_test("FIFO_test");
end
endmodule
```

Fifo_design:

```
module FIFO(fifo_if.DUT fifoif);

parameter FIFO_DEPTH = 8;
localparam max_fifo_addr = $clog2(FIFO_DEPTH);
reg [fifoif.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  //resetting all values to 0
            wr_ptr <= 0;
fifoif.wr_ack <= 0;
fifoif.overflow <= 0;
```

```
        end else if (fifoif.wr_en && count < FIFO_DEPTH) begin
//modified the condition to check if the FIFO is fifoif.full or not
            mem[wr_ptr] <= fifoif.data_in;
            fifoif.wr_ack <= 1;
            wr_ptr <= wr_ptr + 1;
        end else if (fifoif.wr_en && count == FIFO_DEPTH) begin
            fifoif.wr_ack <= 0;   // Acknowledge that write wasn't
successful
            fifoif.overflow <= 1; // Set fifoif.overflow if trying to
write when fifoif.full
        end else begin
            fifoif.wr_ack <= 0;
            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;
    end
    else if (fifoif.rd_en && count != 0) begin
        fifoif.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
    else if (fifoif.rd_en && count == 0) begin
        fifoif.underflow <= 1; // Set fifoif.underflow if trying to read
when fifoif.empty
    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) && count <
FIFO_DEPTH)
            count <= count + 1;
        else if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b01) && count != 0)
```

```
count <= count - 1;
else if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b11) && count ==
FIFO_DEPTH) //read only
count <= count - 1;
else if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b11) && count ==
0)    //write only
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;
//Sequential
assign fifoif.almostfull = (count == FIFO_DEPTH-1)? 1 : 0;     //
Trigger when one slot is left
assign fifoif.almostempty = (count == 1)? 1 : 0;
endmodule
```

Do file:
For simulation:

```
vlib work
vlog -f final_src1.txt
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all
add wave -position insertpoint \
sim:/top/fifoif/almostempty \ sim:/top/fifoif/almostfull
\
sim:/top/fifoif/clk \
sim:/top/fifoif/data_in \
sim:/top/fifoif/data_out \
sim:/top/fifoif/empty \
sim:/top/fifoif/FIFO_WIDTH \
sim:/top/fifoif/full \
sim:/top/fifoif/overflow \
sim:/top/fifoif/rd_en \
sim:/top/fifoif/rst_n \
sim:/top/fifoif/underflow \
```

```
sim:/top/fifoif/wr_ack \
sim:/top/fifoif/wr_en
add wave
/sequence_fifo_pkg::write_only_sequence::body/#ublk#6744381
5#36/immed  40
/sequence_fifo_pkg::read_only_sequence::body/#ublk#67443815
#59/immed  63
/sequence_fifo_pkg::read_write_sequence::body/#ublk#6744381
5#81/immed__85 /top/DUTf/sva1/EMPTY_assertion
/top/DUTf/sva1/ALMOSTEMPTY_assertion
/top/DUTf/sva1/ALMOSTFULL_assertion
/top/DUTf/sva1/FULL_assertion
/top/DUTf/sva1/OVERFLOW_assertion
/top/DUTf/sva1/UNDERFLOW_assertion
/top/DUTf/sva1/WR_ACK_HIGH_assertion
/top/DUTf/sva1/WR_ACK_LOW_assertion
/top/DUTf/sva1/COUNTER_0_assertion
/top/DUTf/sva1/COUNTER_INC_10_assertion
/top/DUTf/sva1/COUNTER_INC_01_assertion
/top/DUTf/sva1/COUNTER_INC_11_WR_assertion
/top/DUTf/sva1/COUNTER_INC_11_RD_assertion
/top/DUTf/sva1/COUNTER_LAT_assertion
/top/DUTf/sva1/PTR_RST_assertion
/top/DUTf/sva1/RD_PTR_assertion
/top/DUTf/sva1/WR_PTR_assertion
run -all
```

For Assertions/Coverage report:

```
vlib work
vlog -f final_src1.txt
vlog -work work -vopt -sv -stats=none
+incdir+path_to_assertions_dir +define+SIM FIFO.sv
vsim -voptargs=+acc work.top -cover
run -all
coverage save top.ucdb -du FIFO -onexit
coverage report -detail -assert -cvg -directive -comments
-output Assertion_Fcoverage_reports1.txt {}
coverage exclude -cvgpath
{/FIFO_Coverage_pkg/FIFO_Coverage/covCode/#cross
0#/<auto[ 0],auto[1],auto[1]>}
```

```
{/FIFO_Coverage_pkg/FIFO_Coverage/covCode/#cross
0#/<auto[ 0],auto[0],auto[1]>}
coverage exclude -cvgpath
{/FIFO_Coverage_pkg/FIFO_Coverage/covCode/#cross
1#/<auto[ 0],auto[1],auto[1]>}
{/FIFO_Coverage_pkg/FIFO_Coverage/covCode/#cross
1#/<auto[ 0],auto[0],auto[1]>}
coverage exclude -cvgpath
{/FIFO_Coverage_pkg/FIFO_Coverage/covCode/#cross
2#/<auto[ 1],auto[1],auto[1]>}
{/FIFO_Coverage_pkg/FIFO_Coverage/covCode/#cross
2#/<auto[ 0],auto[1],auto[1]>}
```

For Code coverage:

```
vlog -f final_src1.txt +cover
vsim -voptargs=+acc work.top -cover
run -all
coverage save top1.ucdb -du FIFO -onexit
quit -sim
vcover report top1.ucdb -details -annotate -all -output
Code_coverage_reports.txt
```