Synchronous FIFO Verification (UVM)

Adnan Bahaaeldin Atef

➢ FIFO Specifications:

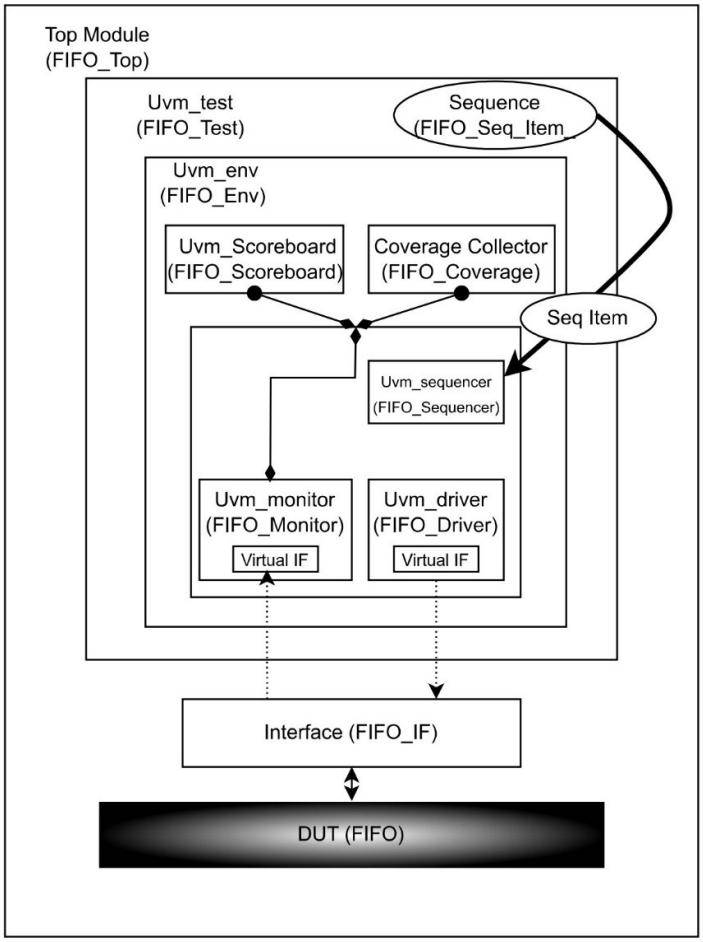
Port	Direction	Function
data_in	Input	Write Data: The input data bus used when writing the FIFO.
wr_en		Write Enable: If the FIFO is not full, asserting this signal causes data (on data_in) to be written into the FIFO
rd_en		Read Enable: If the FIFO is not empty, asserting this signal causes data (on data_out) to be read from the FIFO
clk		Clock signal
rst_n		Active low asynchronous reset
data_out	Output	Read Data: The sequential output data bus used when reading from the FIFO.
full		Full Flag: When asserted, this combinational output signal indicates that the FIFO is full. Write requests are ignored when the FIFO is full, initiating a write when the FIFO is full is not destructive to the contents of the FIFO.
almostfull		Almost Full: When asserted, this combinational output signal indicates that only one more write can be performed before the FIFO is full.
empty		Empty Flag: When asserted, this combinational output signal indicates that the FIFO is empty. Read requests are ignored when the FIFO is empty, initiating a read while empty is not destructive to the FIFO.
almostempty		Almost Empty: When asserted, this output combinational signal indicates that only one more read can be performed before the FIFO goes to empty.
overflow		Overflow: This sequential output signal indicates that a write request (wr_en) was rejected because the FIFO is full. Overflowing the FIFO is not destructive to the contents of the FIFO.

underflow	Output	Underflow: This sequential output signal Indicates that the read request (rd_en) was rejected because the FIFO is empty. Under flowing the FIFO is not destructive to the FIFO.
wr_ack		Write Acknowledge: This sequential output signal indicates that a write request (wr_en) has succeeded.

➢ Verification Plan:

Label	Dsign Requirements Description	Stimulus Generation	Functional Coverage	Functionality check
FIFO_1	When the reset is asserted at the start of TB wr_ptr, rd_ptr, wr_ack, overflow, and count should be deasserted.	Directed at the start of the simulation then randomized with constraint that drive the reset to be off most of the time.	-	Immediate assertion in the top, and concurrent assertion inside the DUT to check that output signals are correctly reset
FIFO_2	When the wr_en signal is asserted and rd_en is deasserted the value inside data_in should be written inside the fifo at wr_ptr position	Directed test cases, where wr_en is asserted, rd_en is deasserted, reset is deasserted and data_in = {0 => 7}	Cross coverage between wr_en, rd_en and wr_ack, overflow and full signals	data_out is checked against scoreboard output referenece and other output signals are checked using SVA
FIFO_3	When both rd_en and wr_en are asserted only read should take place when the fifo is full	Directed test cases, where wr_en is asserted, rd_en is asserted, reset is deasserted.	Cover all values of ouput data signals and Cross coverage between wr_en, rd_en and all output data signals	data_out is checked against scoreboard output referenece and other output signals are checked using SVA
FIFO_4	When the rd_en signal is asserted and wr_en is deasserted the value inside the fifo at rd_ptr position should be read correctly	Directed test cases, where wr_en is deasserted, rd_en is asserted, reset is deasserted.	Cross coverage between wr_en, rd_en and underflow, empty and almost_empty signals	data_out is checked against scoreboard output referenece and other output signals are checked using SVA
FIFO_5	When reading while the fifo is empty the operation isn't destructive to the fifo	Directed test cases, where wr_en is deasserted, rd_en is asserted, reset is deasserted.	r_en, rd_en and underflow, empty	data_out is checked against scoreboard output referenece and other output signals are checked using SVA
FIFO_6	When both rd_en and wr_en are asserted only write should take place when the fifo is empty	Directed test cases, where wr_en is asserted, rd_en is asserted, reset is deasserted.	Cover all values of ouput data signals and Cross coverage between wr_en, rd_en and all output data signals	data_out is checked against scoreboard output referenece and other output signals are checked using SVA
FIFO_7	When writing for more times than fifo size allows the write operation must not be destructive to contents of fifo	Directed test cases, where wr_en is asserted, rd_en is deasserted, reset is deasserted and data_in = {0 => 10}	Cross coverage between wr_en, rd_en and wr_ack, overflow and full signals	data_out is checked against scoreboard output referenece and other output signals are checked using SVA
FIFO_8	When multiple simultaneous/concurrent read, write and reset operations are done the fifo should give valid and stable outputs	Randomization for 10000 times under constraints that rst_n be enabled 5% of the time, wr_en be enabled 30% and rd_en be enabled 70% of the time.	Cover all values of ouput data signals and Cross coverage between wr_en, rd_en and all output data signals	data_out is checked against scoreboard output referenece and other output signals are checked using SVA

➢ UVM_Environment:



➢ UVM flow summary:

1. The top module instantiates the interface and the DUT and bind the assertions, it's also where the clock is created.
2. The top module passes the virtual interface using the configuration database and runs the Test.
3. Uvm_test retrieves the virtual interface from the database and creates a configuration object and assigns that virtual interface to the one inside the configuration object.
4. Uvm_test then sets the configuration object inside the database. And then starts the sequences on the sequencer.

5. Uvm_env then builds and connects the agent and analysis components (Scoreboard and Coverage collector).
6. Uvm_Agent builds and connects the sequencer, driver and monitor.
7. The Sequencer sends the sequence items to the driver for execution.
8. The Driver pulls the sequence item from the sequencer.
9. The Driver drives the sequence item inside the run_phase task using the virtual interface.
10. Uvm_monitor captures signal information from the DUT ports where it'll send the information to analysis components by creating a sequence item.
11. Uvm_scoreboard gets the sequence item from the monitor and runs the reference model function to compare the DUT output with the expected output.
12. Finally, the coverage collector also receives the sequence item from the monitor and samples the data fields for functional coverage.

➢ Design Code:

```
1.  ////////////////////////////////////////////////////////////////////////////////
2.  // Author: Kareem Waseem
3.  // Course: Digital Verification using SV & UVM
4.  //
5.  // Description: FIFO Design
6.  //
7.  ////////////////////////////////////////////////////////////////////////////////
8.
9.  import FIFO_Shared_pkg::*; // Importing shared parameters
10.
11. module FIFO(FIFO_IF.DUT fifo_if);
12.
13. localparam max_fifo_addr = $clog2(FIFO_DEPTH);
14.
15. reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];
16.
17. reg [max_fifo_addr-1:0] wr_ptr =0, rd_ptr=0;
18. reg [max_fifo_addr:0] count=0;
19.
20. always @(posedge fifo_if.clk or negedge fifo_if.rst_n) begin
21.     if (!fifo_if.rst_n) begin
22.         wr_ptr <= 0;
23.         fifo_if.overflow <= 0;
24.         fifo_if.wr_ack <= 0;
25.     end
26.     else if (fifo_if.wr_en && !fifo_if.rd_en && count < FIFO_DEPTH) begin
27.         mem[wr_ptr] <= fifo_if.data_in;
28.         fifo_if.wr_ack <= 1;
29.         wr_ptr <= wr_ptr + 1;
30.         fifo_if.overflow <= 0;
31.     end
32.     else begin
33.         if (fifo_if.full && fifo_if.wr_en) begin
34.             fifo_if.wr_ack <= 0;
35.             fifo_if.overflow <= 1;
36.         end
37.         else begin
38.             fifo_if.overflow <= 0;
39.         end
40.     end
41. end
42.
43. always @(posedge fifo_if.clk or negedge fifo_if.rst_n) begin
44.     if (!fifo_if.rst_n) begin
45.         rd_ptr <= 0;
46.     end
47.     else if (fifo_if.rd_en && !fifo_if.wr_en && count != 0) begin
48.         fifo_if.data_out <= mem[rd_ptr];
49.         rd_ptr <= rd_ptr + 1;
50.     end
51. end
```

```
52.
53.    always @(posedge fifo_if.clk or negedge fifo_if.rst_n) begin
54.        if (!fifo_if.rst_n) begin
55.            count <= 0;
56.        end
57.        else begin
58.            if  ( ({fifo_if.wr_en, fifo_if.rd_en} == 2'b10) && !fifo_if.full)
59.                count <= count + 1;
60.            else if ( ({fifo_if.wr_en, fifo_if.rd_en} == 2'b01) && !fifo_if.empty)
61.                count <= count - 1;
62.        end
63.    end
64.
65.    //handle case where both wr_en and rd_en are high
66.    always @(posedge fifo_if.clk) begin
67.        if(fifo_if.wr_en && fifo_if.rd_en) begin
68.
69.                //If FIFO is empty, only write is allowed
70.                if(count == 0) begin
71.                    mem[wr_ptr] = fifo_if.data_in;
72.                    fifo_if.wr_ack = 1;
73.                    wr_ptr = wr_ptr +1;
74.                    count++;
75.                end
76.
77.                //If FIFO is full, only read is allowed
78.                else if(count >= FIFO_DEPTH) begin
79.                    fifo_if.data_out = mem[rd_ptr];
80.                    rd_ptr = rd_ptr +1;
81.                    count--;
82.                end
83.
84.                //Read and write
85.                else begin
86.                    mem[wr_ptr] = fifo_if.data_in;
87.                    fifo_if.wr_ack = 1;
88.                    wr_ptr = wr_ptr +1;
89.                    fifo_if.data_out = mem[rd_ptr];
90.                    rd_ptr = rd_ptr + 1;
91.                end
92.        end
93.    end
94.
95.    assign fifo_if.full = (count == FIFO_DEPTH)? 1 : 0;
96.    assign fifo_if.empty = (count == 0)? 1 : 0;
97.    assign fifo_if.underflow = (fifo_if.empty && fifo_if.rd_en)? 1 : 0;
98.
99.    assign fifo_if.almostfull = (count == FIFO_DEPTH-1)? 1 : 0;
100.
101.   assign fifo_if.almostempty = (count == 1)? 1 : 0;
102.
103.   endmodule
```

➢ List of bugs found inside design:

1. ***Bug_1:*** Overflow wasn't set to zero on reset.

   a. Before Fixing:

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        wr_ptr <= 0;
    end
```

   b. After Fixing:

```
always @(posedge fifo_if.clk or negedge fifo_if.rst_n) begin
    if (!fifo_if.rst_n) begin
        wr_ptr <= 0;
        fifo_if.overflow <= 0; //Bug_1
```

2. ***Bug_2:*** write acknowledge wasn't set to zero on rest.

   a. Before Fixing:

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        wr_ptr <= 0;
    end
```

   b. After Fixing:

```
always @(posedge fifo_if.clk or negedge fifo_if.rst_n) begin
    if (!fifo_if.rst_n) begin
        wr_ptr <= 0;
        fifo_if.wr_ack <= 0; //Bug_2
    end
```

3. ***Bug_3:*** wr_ack should be set to 0 only when fifo is full and wr_en is asserted.

   a. *Before Fixing:*

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

   b. *After Fixing:*

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

4. ***Bug_4:*** Case where both rd_en and wr_en were asserted wasn't handled correctly

   a. *Before Fixing:*

   i. . rd_en signal wasn't checked to be low when writing.

```
else if (wr_en && count < FIFO_DEPTH) begin
    mem[wr_ptr] <= data_in;
    wr_ack <= 1;
    wr_ptr <= wr_ptr + 1;
end
```

   ii. wr_en wasn't checked to be low on read operations

```
else if (rd_en && count != 0) begin
    data_out <= mem[rd_ptr];
    rd_ptr <= rd_ptr + 1;
end
```

b. <u>After Fixing:</u>

i. rd_en signal is checked to be low when writing.

```
else if (fifo_if.wr_en && !fifo_if.rd_en && count < FIFO_DEPTH) begin
    mem[wr_ptr] <= fifo_if.data_in;
    fifo_if.wr_ack <= 1;
    wr_ptr <= wr_ptr + 1;
    fifo_if.overflow <= 0;
end
```

ii. wr_en is checked to be low on read operations

```
else if (fifo_if.rd_en && !fifo_if.wr_en && count != 0) begin
    fifo_if.data_out <= mem[rd_ptr];
    rd_ptr <= rd_ptr + 1;
end
```

iii. <u>Handled case where both wr_en and rd_en are high correctly.</u>

```
//handle case where both wr_en and rd_en are high
always @(posedge fifo_if.clk) begin
    if(fifo_if.wr_en && fifo_if.rd_en) begin

            //If FIFO is empty, only write is allowed
            if(count == 0) begin
                mem[wr_ptr] = fifo_if.data_in;
                fifo_if.wr_ack = 1;
                wr_ptr = wr_ptr +1;
                count++;
            end

            //If FIFO is full, only read is allowed
            else if(count >= FIFO_DEPTH) begin
                fifo_if.data_out = mem[rd_ptr];
                rd_ptr = rd_ptr +1;
                count--;
            end

            //Read and write
            else begin
                mem[wr_ptr] = fifo_if.data_in;
                fifo_if.wr_ack = 1;
                wr_ptr = wr_ptr +1;
                fifo_if.data_out = mem[rd_ptr];
                rd_ptr = rd_ptr + 1;
            end
    end
end
```

5. ***Bug_5:*** Almost full should be checked against FIFO_DEPTH-1 not FIFO_DEPTH-2.

   a. *Before Fixing:*

```
assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
```

   b. *After Fixing:*

```
assign fifo_if.almostfull = (count == FIFO_DEPTH-1)? 1 : 0; //BUG_5: FIFO_DEPTH-1
```

➢ Code Coverage:

```
================================================================================
=== File: FIFO.sv
================================================================================
Statement Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           37        37         0     100.0
```

```
Branch Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Branches                        28        28         0     100.0
```

```
Condition Coverage:
    Enabled Coverage            Active    Covered    Misses % Covered
    ----------------            ------      ----    ------ ---------
    FEC Condition Terms              18        18         0     100.0
```

```
================================Toggle Details================================

Toggle Coverage for File FIFO.sv --

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

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

Toggle Coverage      =      100.0% (20 of 20 bins)
```

➢ Functional Coverage:

```
COVERGROUP COVERAGE:
----------------------------------------------------------------------------------------------------
Covergroup                                              Metric       Goal    Status

----------------------------------------------------------------------------------------------------
 TYPE /FIFO_Coverage_pkg/FIFO_Coverage/cover_group     100.0%        100    Covered
    covered/total bins:                                    66         66
    missing/total bins:                                     0         66
    % Hit:                                             100.0%        100
    Coverpoint cover_group::WRITE_ENABLE               100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                      7030          1    Covered
        bin auto[1]                                      3003          1    Covered
    Coverpoint cover_group::READ_ENABLE                100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                      3003          1    Covered
        bin auto[1]                                      7030          1    Covered
    Coverpoint cover_group::WRITE_ACK                  100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                      1570          1    Covered
        bin auto[1]                                      8465          1    Covered
    Coverpoint cover_group::OVERFLOW                   100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                     10027          1    Covered
        bin auto[1]                                         8          1    Covered
    Coverpoint cover_group::UNDERFLOW                  100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                      5299          1    Covered
        bin auto[1]                                      4734          1    Covered
```

```
    Coverpoint cover_group::FULL                       100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                     10025          1    Covered
        bin auto[1]                                        10          1    Covered
    Coverpoint cover_group::EMPTY                      100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                      3923          1    Covered
        bin auto[1]                                      6112          1    Covered
    Coverpoint cover_group::ALMOSTFULL                 100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                     10032          1    Covered
        bin auto[1]                                         3          1    Covered
    Coverpoint cover_group::ALMOSTEMPTY                100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin auto[0]                                      6775          1    Covered
        bin auto[1]                                      3260          1    Covered
    Cross cover_group::WRITE_READ_ACK_CROSS            100.0%        100    Covered
        covered/total bins:                                 8          8
        missing/total bins:                                 0          8
        % Hit:                                         100.0%        100
        bin <auto[0],auto[0],auto[0]>                     436          1    Covered
        bin <auto[1],auto[0],auto[0]>                      54          1    Covered
        bin <auto[0],auto[1],auto[0]>                     970          1    Covered
        bin <auto[1],auto[1],auto[0]>                     108          1    Covered
        bin <auto[0],auto[0],auto[1]>                    1651          1    Covered
        bin <auto[1],auto[0],auto[1]>                     862          1    Covered
        bin <auto[0],auto[1],auto[1]>                    3973          1    Covered
        bin <auto[1],auto[1],auto[1]>                    1979          1    Covered
```

```
    Cross cover_group::WRITE_READ_OVERFLOW_CROSS       100.0%        100    Covered
        covered/total bins:                                6          6
        missing/total bins:                                0          6
        % Hit:                                         100.0%        100
        bin <auto[0],auto[0],auto[0]>                   2087          1    Covered
        bin <auto[0],auto[1],auto[0]>                   4943          1    Covered
        bin <auto[1],auto[0],auto[0]>                    909          1    Covered
        bin <auto[1],auto[1],auto[0]>                   2086          1    Covered
        bin <auto[1],auto[0],auto[1]>                      7          1    Covered
        bin <auto[1],auto[1],auto[1]>                      1          1    Covered
        ignore_bin bins_ignore1                            0               ZERO
        ignore_bin bins_ignore2                            0               ZERO
    Cross cover_group::WRITE_READ_UNDERFLOW_CROSS      100.0%        100    Covered
        covered/total bins:                                6          6
        missing/total bins:                                0          6
        % Hit:                                         100.0%        100
        bin <auto[0],auto[0],auto[0]>                   2087          1    Covered
        bin <auto[1],auto[0],auto[0]>                    916          1    Covered
        bin <auto[0],auto[1],auto[0]>                    316          1    Covered
        bin <auto[1],auto[1],auto[0]>                   1980          1    Covered
        bin <auto[0],auto[1],auto[1]>                   4627          1    Covered
        bin <auto[1],auto[1],auto[1]>                    107          1    Covered
        ignore_bin bins_ignore1                            0               ZERO
        ignore_bin bins_ignore2                            0               ZERO
    Cross cover_group::WRITE_READ_FULL_CROSS           100.0%        100    Covered
        covered/total bins:                                6          6
        missing/total bins:                                0          6
        % Hit:                                         100.0%        100
        bin <auto[0],auto[0],auto[0]>                   2086          1    Covered
        bin <auto[1],auto[0],auto[0]>                    907          1    Covered
        bin <auto[0],auto[0],auto[1]>                      1          1    Covered
        bin <auto[1],auto[0],auto[1]>                      9          1    Covered
        bin <auto[0],auto[1],auto[0]>                   4943          1    Covered
        bin <auto[1],auto[1],auto[0]>                   2087          1    Covered
        ignore_bin bins_ignore1                            0               ZERO
        ignore_bin bins_ignore2                            0               ZERO
```

```
    Cross cover_group::WRITE_READ_EMPTY_CROSS          100.0%        100    Covered
        covered/total bins:                                8          8
        missing/total bins:                                0          8
        % Hit:                                         100.0%        100
        bin <auto[0],auto[0],auto[0]>                    758          1    Covered
        bin <auto[1],auto[0],auto[0]>                    869          1    Covered
        bin <auto[0],auto[1],auto[0]>                    316          1    Covered
        bin <auto[1],auto[1],auto[0]>                   1980          1    Covered
        bin <auto[0],auto[0],auto[1]>                   1329          1    Covered
        bin <auto[1],auto[0],auto[1]>                     47          1    Covered
        bin <auto[0],auto[1],auto[1]>                   4627          1    Covered
        bin <auto[1],auto[1],auto[1]>                    107          1    Covered
    Cross cover_group::WRITE_READ_ALMOSTFULL_CROSS     100.0%        100    Covered
        covered/total bins:                                6          6
        missing/total bins:                                0          6
        % Hit:                                         100.0%        100
        bin <auto[0],auto[0],auto[0]>                   2087          1    Covered
        bin <auto[0],auto[1],auto[0]>                   4943          1    Covered
        bin <auto[1],auto[0],auto[0]>                    914          1    Covered
        bin <auto[1],auto[1],auto[0]>                   2086          1    Covered
        bin <auto[1],auto[0],auto[1]>                      2          1    Covered
        bin <auto[1],auto[1],auto[1]>                      1          1    Covered
        ignore_bin bins_ignore1                            0               ZERO
        ignore_bin bins_ignore2                            0               ZERO
    Cross cover_group::WRITE_READ_ALMOSTEMPTY_CROSS
                                                       100.0%        100    Covered
        covered/total bins:                                8          8
        missing/total bins:                                0          8
        % Hit:                                         100.0%        100
        bin <auto[0],auto[0],auto[0]>                   1457          1    Covered
        bin <auto[1],auto[0],auto[0]>                    412          1    Covered
        bin <auto[0],auto[1],auto[0]>                   4680          1    Covered
        bin <auto[1],auto[1],auto[0]>                    224          1    Covered
        bin <auto[0],auto[0],auto[1]>                    630          1    Covered
        bin <auto[1],auto[0],auto[1]>                    504          1    Covered
        bin <auto[0],auto[1],auto[1]>                    263          1    Covered
        bin <auto[1],auto[1],auto[1]>                   1863          1    Covered
 CLASS FIFO_Coverage

TOTAL COVERGROUP COVERAGE: 100.0%  COVERGROUP TYPES: 1
```

➢ Assertion Coverage:

```
DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Count Status
                                        Unit   UnitType
--------------------------------------------------------------------------------------------
/top/dut/fifo_inst/cover__fifo_ptr_check FIFO_Assertions Verilog  SVA  FIFO_Assertions.sv(111)
                                                                          10035 Covered
/top/dut/fifo_inst/cover__count_wraparound_check
                                        FIFO_Assertions Verilog  SVA  FIFO_Assertions.sv(102)
                                                                              1 Covered
/top/dut/fifo_inst/cover__write_ptr_wraparound_check
                                        FIFO_Assertions Verilog  SVA  FIFO_Assertions.sv(93)
                                                                            174 Covered
/top/dut/fifo_inst/cover__read_ptr_wraparound_check
                                        FIFO_Assertions Verilog  SVA  FIFO_Assertions.sv(84)
                                                                            157 Covered
/top/dut/fifo_inst/cover__fifo_almostempty_check
                                        FIFO_Assertions Verilog  SVA  FIFO_Assertions.sv(75)
                                                                           3108 Covered
/top/dut/fifo_inst/cover__fifo_almostfull_check
                                        FIFO_Assertions Verilog  SVA  FIFO_Assertions.sv(66)
                                                                              3 Covered
/top/dut/fifo_inst/cover__fifo_full_check
                                        FIFO_Assertions Verilog  SVA  FIFO_Assertions.sv(57)
                                                                              9 Covered
/top/dut/fifo_inst/cover__fifo_empty_check
                                        FIFO_Assertions Verilog  SVA  FIFO_Assertions.sv(48)
                                                                           5771 Covered
/top/dut/fifo_inst/cover__fifo_underflow_check
                                        FIFO_Assertions Verilog  SVA  FIFO_Assertions.sv(39)
                                                                           4052 Covered
/top/dut/fifo_inst/cover__fifo_overflow_check
                                        FIFO_Assertions Verilog  SVA  FIFO_Assertions.sv(31)
                                                                              8 Covered
/top/dut/fifo_inst/cover__fifo_write_check
                                        FIFO_Assertions Verilog  SVA  FIFO_Assertions.sv(23)
                                                                           2711 Covered
/top/dut/fifo_inst/cover__reset_check   FIFO_Assertions Verilog  SVA  FIFO_Assertions.sv(15)
                                                                            525 Covered

TOTAL DIRECTIVE COVERAGE: 100.0%  COVERS: 12
```

```
ASSERTION RESULTS:
-----------------------------------------------------
Name                 File(Line)           Failure Pass
                                          Count   Count
-----------------------------------------------------
/top/dut/fifo_inst/check_reset_assertion
                     FIFO_Assertions.sv(14)       0     1
/top/dut/fifo_inst/check_fifo_write_assertion
                     FIFO_Assertions.sv(22)       0     1
/top/dut/fifo_inst/check_fifo_overflow_assertion
                     FIFO_Assertions.sv(30)       0     1
/top/dut/fifo_inst/check_fifo_underflow_assertion
                     FIFO_Assertions.sv(38)       0     1
/top/dut/fifo_inst/check_fifo_empty_assertion
                     FIFO_Assertions.sv(47)       0     1
/top/dut/fifo_inst/check_fifo_full_assertion
                     FIFO_Assertions.sv(56)       0     1
/top/dut/fifo_inst/check_fifo_almostfull_assertion
                     FIFO_Assertions.sv(65)       0     1
/top/dut/fifo_inst/check_fifo_almostempty_assertion
                     FIFO_Assertions.sv(74)       0     1
/top/dut/fifo_inst/check_read_ptr_wraparound_assertion
                     FIFO_Assertions.sv(83)       0     1
/top/dut/fifo_inst/check_write_ptr_wraparound_assertion
                     FIFO_Assertions.sv(92)       0     1
/top/dut/fifo_inst/check_count_wraparound_assertion
                     FIFO_Assertions.sv(101)       0     1
/top/dut/fifo_inst/check_fifo_ptr_assertion
                     FIFO_Assertions.sv(110)       0     1
/FIFO_Random_Sequence_pkg/FIFO_Random_Sequence/body/#ublk#217107911#18/immed__23
                     FIFO_Random_Sequence.sv(23)       0     1
```

➢ Assertions Code:

```
1.  import FIFO_Sequence_Item_pkg::*;
2.  import FIFO_Shared_pkg::*;
3.  import uvm_pkg::*;
4.  module FIFO_Assertions(FIFO_IF.DUT fifo_if);
5.
6.  //Assertions to check FIFO functionality
7.
8.  property reset_check;
9.  @(posedge fifo_if.clk)
10. (!fifo_if.rst_n) |=> ((dut.rd_ptr == 1'b0) && (dut.wr_ptr  == 1'b0) && (dut.count
    == 1'b0) && (fifo_if.wr_ack == 0) && (fifo_if.overflow == 0));
11. endproperty
12.
13. check_reset_assertion:assert property (reset_check) else $error("Assertion
    reset_check failed!");
14. cover property (reset_check);
15.
16. property fifo_write_check;
17. @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
18. ((fifo_if.wr_en==1'b1) && (fifo_if.full == 1'b0)) |=> (fifo_if.wr_ack == 1'b1);
19. endproperty
20.
21. check_fifo_write_assertion: assert property (fifo_write_check) else
    $error("Assertion fifo_write_check failed!");
22. cover property (fifo_write_check);
23.
24. property fifo_overflow_check;
25. @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
26. ((fifo_if.wr_en == 1'b1) && (fifo_if.full == 1'b1)) |=> (fifo_if.overflow ==
    1'b1);
27. endproperty
28.
29. check_fifo_overflow_assertion: assert property (fifo_overflow_check) else
    $error("Assertion fifo_overflow_check failed!");
30. cover property (fifo_overflow_check);
31.
32. property fifo_underflow_check;
33. @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
34. ((fifo_if.rd_en == 1'b1) && (fifo_if.empty == 1'b1)) |-> (fifo_if.underflow ==
    1'b1);
35. endproperty
36.
37. check_fifo_underflow_assertion: assert property (fifo_underflow_check) else
    $error("Assertion fifo_underflow_check failed!");
38. cover property (fifo_underflow_check);
39.
40. property fifo_empty_check;
41. @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
42. (dut.count == 0) |-> (fifo_if.empty == 1'b1);
```

```
43.    endproperty
44.
45.    check_fifo_empty_assertion: assert property (fifo_empty_check) else
       $error("Assertion fifo_empty_check failed!");
46.    cover property (fifo_empty_check);
47.
48.    property fifo_full_check;
49.    @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
50.    (dut.count == FIFO_DEPTH) |-> (fifo_if.full == 1'b1);
51.    endproperty
52.
53.    check_fifo_full_assertion: assert property (fifo_full_check) else
       $error("Assertion fifo_full_check failed!");
54.    cover property (fifo_full_check);
55.
56.    property fifo_almostfull_check;
57.    @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
58.    (dut.count == FIFO_DEPTH-1) |-> (fifo_if.almostfull == 1'b1);
59.    endproperty
60.
61.    check_fifo_almostfull_assertion: assert property (fifo_almostfull_check) else
       $error("Assertion fifo_almostfull_check failed!");
62.    cover property (fifo_almostfull_check);
63.
64.    property fifo_almostempty_check;
65.    @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
66.    (dut.count == 1) |-> (fifo_if.almostempty == 1'b1);
67.    endproperty
68.
69.    check_fifo_almostempty_assertion: assert property (fifo_almostempty_check) else
       $error("Assertion fifo_almostempty_check failed!");
70.    cover property (fifo_almostempty_check);
71.
72.    property read_ptr_wraparound_check;
73.    @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
74.    ((dut.rd_ptr == (FIFO_DEPTH-1)) && (fifo_if.rd_en == 1'b1) && (fifo_if.empty ==
       1'b0)) |=> (dut.rd_ptr == 0);
75.    endproperty
76.
77.    check_read_ptr_wraparound_assertion: assert property (read_ptr_wraparound_check)
       else $error("Assertion read_ptr_wraparound_check failed!");
78.    cover property (read_ptr_wraparound_check);
79.
80.    property write_ptr_wraparound_check;
81.    @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n)
82.    ((dut.wr_ptr == (FIFO_DEPTH-1)) && (fifo_if.wr_en == 1'b1) && (fifo_if.full ==
       1'b0)) |=> (dut.wr_ptr == 0);
83.    endproperty
84.
85.    check_write_ptr_wraparound_assertion: assert property
       (write_ptr_wraparound_check) else $error("Assertion write_ptr_wraparound_check
       failed!");
86.    cover property (write_ptr_wraparound_check);
87.
```

```
88.   property count_wraparound_check;
89.   @(posedge fifo_if.clk)
90.   ((fifo_if.rst_n == 1'b0) && ($past(fifo_if.rst_n) == 1'b1) && ($past(dut.count)
      == 8)) |=> dut.count == 0;
91.   endproperty
92.
93.   check_count_wraparound_assertion: assert property (count_wraparound_check) else
      $error("Assertion count_wraparound_check failed!");
94.   cover property (count_wraparound_check);
95.
96.   property fifo_ptr_check;
97.   @(posedge fifo_if.clk)
98.   ((dut.rd_ptr < FIFO_DEPTH) && (dut.wr_ptr < FIFO_DEPTH));
99.   endproperty
100.
101.  check_fifo_ptr_assertion: assert property (fifo_ptr_check) else $error("Assertion
      fifo_ptr_check failed!");
102.  cover property (fifo_ptr_check);
103.
104.  endmodule
```

➢ List of assertions:

Feature	Assertion
When the reset is active all of rd_ptr, wr_ptr, count, wr_ack and overflow should be low.	@(posedge fifo_if.clk) (!fifo_if.rst_n) |=> ((dut.rd_ptr == 1'b0) && (dut.wr_ptr == 1'b0) && (dut.count == 1'b0) && (fifo_if.wr_ack == 0) && (fifo_if.overflow == 0));
Whenever wr_en is high and fifo isn't full, wr_akc is high.	@(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) ((fifo_if.wr_en==1'b1) && (fifo_if.full == 1'b0)) |=> (fifo_if.wr_ack == 1'b1);
Whenever wr_en is high and fifo is full, overflow signal is high.	@(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) ((fifo_if.wr_en == 1'b1) && (fifo_if.full == 1'b1)) |=> (fifo_if.overflow == 1'b1);
Whenever rd_en is high and fifo is empty, underflow signal is high.	@(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) ((fifo_if.rd_en == 1'b1) && (fifo_if.empty == 1'b1)) |-> (fifo_if.underflow == 1'b1);
Whenever count is zero, empty signal should be high.	@(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (dut.count == 0) |-> (fifo_if.empty == 1'b1);
Whenever count is FIFO_DEPTH, full signal should be high.	@(posedge fifo_if.clk) disable iff !fifo_if.rst_n) (dut.count == FIFO_DEPTH) |-> (fifo_if.full == 1'b1);

Whenever count is (FIFO_DEPTH-1), almost full signal should be high.	@(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (dut.count == FIFO_DEPTH-1) |-> (fifo_if.almostfull == 1'b1);
Whenever count is 1, almost empty signals should be high.	@(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (dut.count == 1) |-> (fifo_if.almostempty == 1'b1);
Whenever rd_ptr is (FIFO_DEPTH-1) ,rd_en is high and fifo isn't empty, rd_ptr should wraparound to 0.	@(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) ((dut.rd_ptr == (FIFO_DEPTH-1)) && (fifo_if.rd_en == 1'b1) && (fifo_if.empty == 1'b0)) |=> (dut.rd_ptr == 0);
Whenever wr_ptr is (FIFO_DEPTH-1) ,wr_en is high and fifo isn't full, wr_ptr should wraparound to 0.	@(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) ((dut.wr_ptr == (FIFO_DEPTH-1)) && (fifo_if.wr_en == 1'b1) && (fifo_if.full == 1'b0)) |=> (dut.wr_ptr == 0);
Whenever reset is active and count was FIFO_DEPTH, count should be reset to 0.	@(posedge fifo_if.clk) ((fifo_if.rst_n == 1'b0) && ($past(fifo_if.rst_n) == 1'b1) && ($past(dut.count) == 8)) |=> dut.count == 0;
rd_ptr and wr_ptr should never exceed FIFO_DEPTH at any given time.	@(posedge fifo_if.clk) ((dut.rd_ptr < FIFO_DEPTH) && (dut.wr_ptr < FIFO_DEPTH));

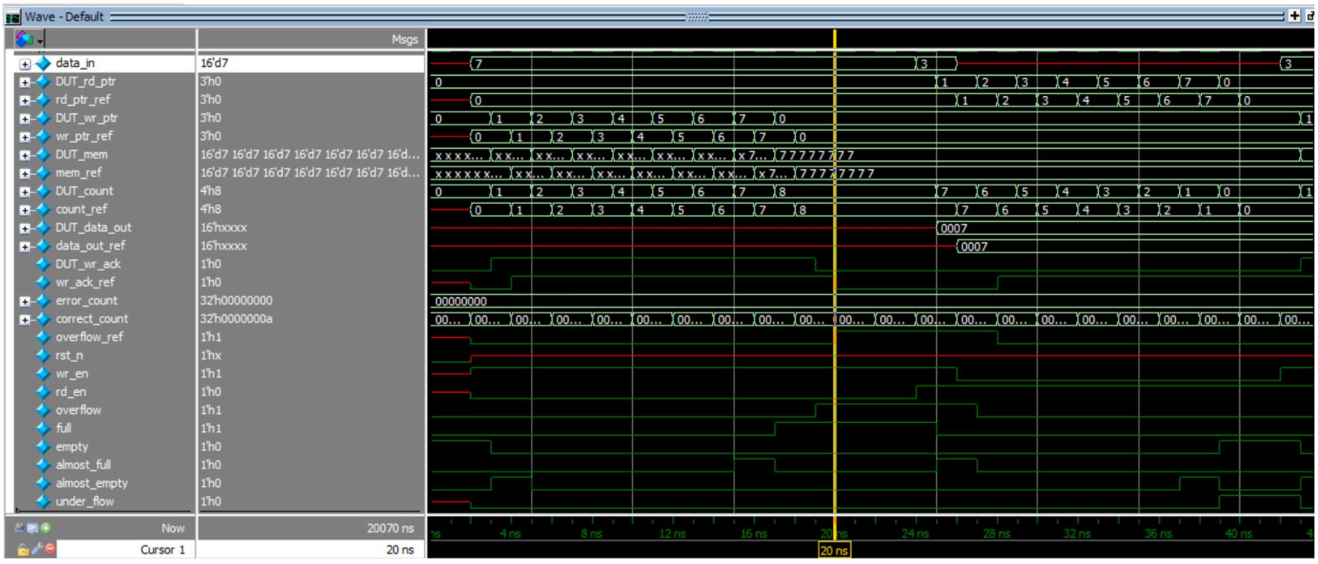
➢ Sequences:

○ Reset Sequence:

```
1.    package FIFO_Reset_Sequence_pkg;
2.    import uvm_pkg::*;
3.    import FIFO_Sequence_Item_pkg::*;
4.
5.    `include "uvm_macros.svh"
6.
7.    class FIFO_Reset_Sequence extends uvm_sequence #(FIFO_Sequence_Item);
8.    `uvm_object_utils(FIFO_Reset_Sequence)
9.
10.   FIFO_Sequence_Item fifo_sequence_item;
11.
12.   function new (string name = "FIFO_Reset_Sequence");
13.       super.new(name);
14.   endfunction
15.
16.   task body();
17.
18.       fifo_sequence_item =
      FIFO_Sequence_Item::type_id::create("fifo_sequence_item");
19.       start_item(fifo_sequence_item);
20.       fifo_sequence_item.rst_n = 0;
21.       finish_item(fifo_sequence_item);
22.       fifo_sequence_item.rst_n = 1;
23.   endtask
24.
25.   endclass
26.
27.   endpackage
```

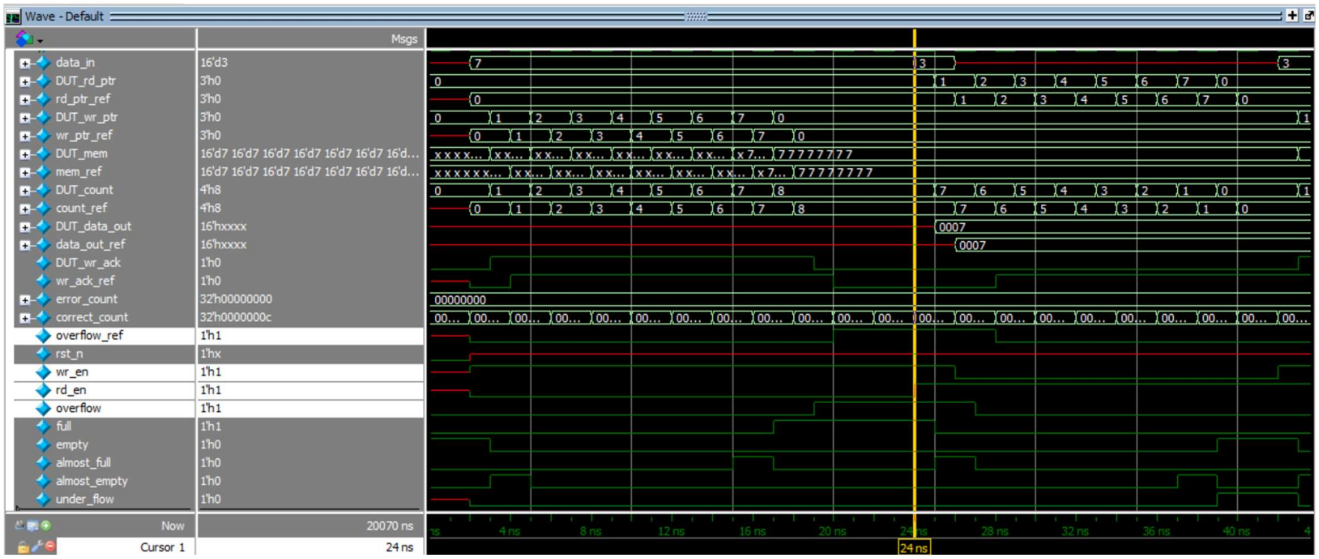
➢ Write Only Loop Sequence:

```
1.  package FIFO_WriteOnly_Loop_Sequence_pkg;
2.
3.  import uvm_pkg::*;
4.  import FIFO_Sequence_Item_pkg::*;
5.  `include "uvm_macros.svh"
6.
7.  class FIFO_WriteOnly_Loop_Sequence extends uvm_sequence
    #(FIFO_Sequence_Item);
8.  `uvm_object_utils(FIFO_WriteOnly_Loop_Sequence)
9.
10. FIFO_Sequence_Item fifo_sequence_item;
11.
12. function new (string name = "FIFO_WriteOnly_Loop_Sequence");
13.     super.new(name);
14. endfunction
15.
16. task body();
17.     repeat(11) begin
18.         fifo_sequence_item =
    FIFO_Sequence_Item::type_id::create("fifo_sequence_item");
19.         start_item(fifo_sequence_item);
20.         fifo_sequence_item.rd_en = 0; //Force rd_en to 0
21.         fifo_sequence_item.data_in = 7; //fill fifo with value 7
22.         fifo_sequence_item.wr_en = 1; //Force wr_en to 1
23.         finish_item(fifo_sequence_item);
24.     end
25.
26. endtask
27. endclass
28.
29. endpackage
30.
```



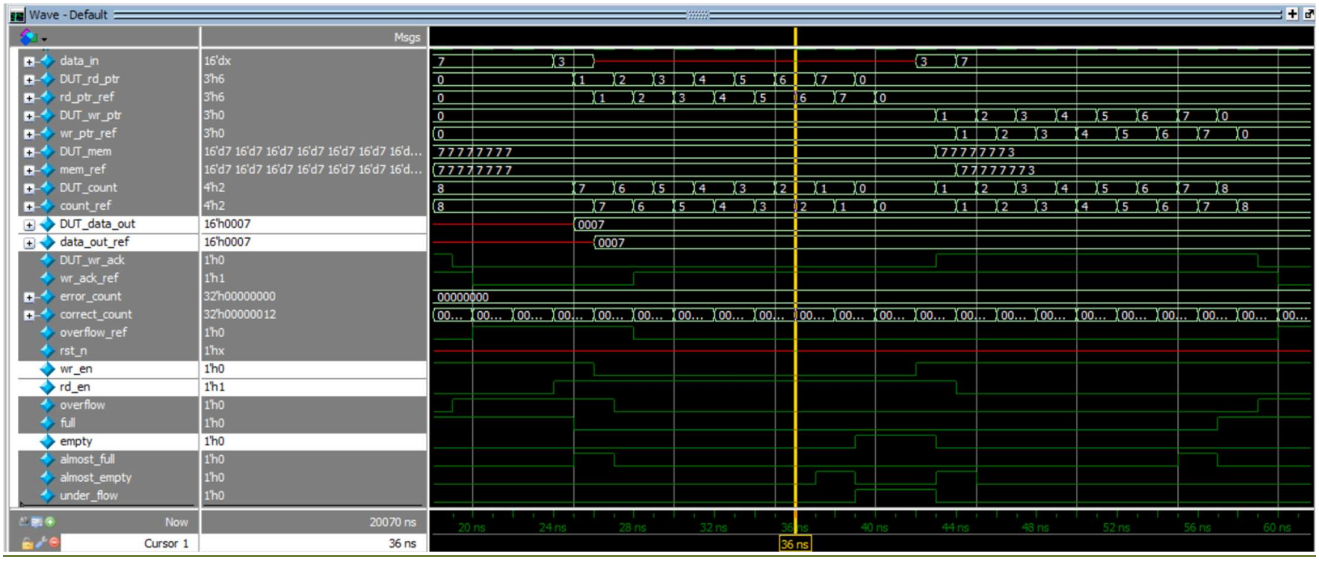
➢ Read Write Sequence:

```
1.    package FIFO_ReadWrite_Sequence_pkg;
2.
3.    import uvm_pkg::*;
4.    import FIFO_Sequence_Item_pkg::*;
5.    `include "uvm_macros.svh"
6.
7.    class FIFO_ReadWrite_Sequence extends uvm_sequence #(FIFO_Sequence_Item);
8.    `uvm_object_utils(FIFO_ReadWrite_Sequence)
9.
10.   FIFO_Sequence_Item fifo_sequence_item;
11.
12.   function new (string name = "FIFO_ReadWrite_Sequence");
13.       super.new(name);
14.   endfunction
15.
16.   task body();
17.       fifo_sequence_item =
      FIFO_Sequence_Item::type_id::create("fifo_sequence_item");
18.       start_item(fifo_sequence_item);
19.       fifo_sequence_item.rd_en = 1; //Force rd_en to 1
20.       fifo_sequence_item.wr_en = 1; //Force wr_en to 1
21.       fifo_sequence_item.data_in = 3;
22.       finish_item(fifo_sequence_item);
23.
24.   endtask
25.   endclass
26.   endpackage
27.
```



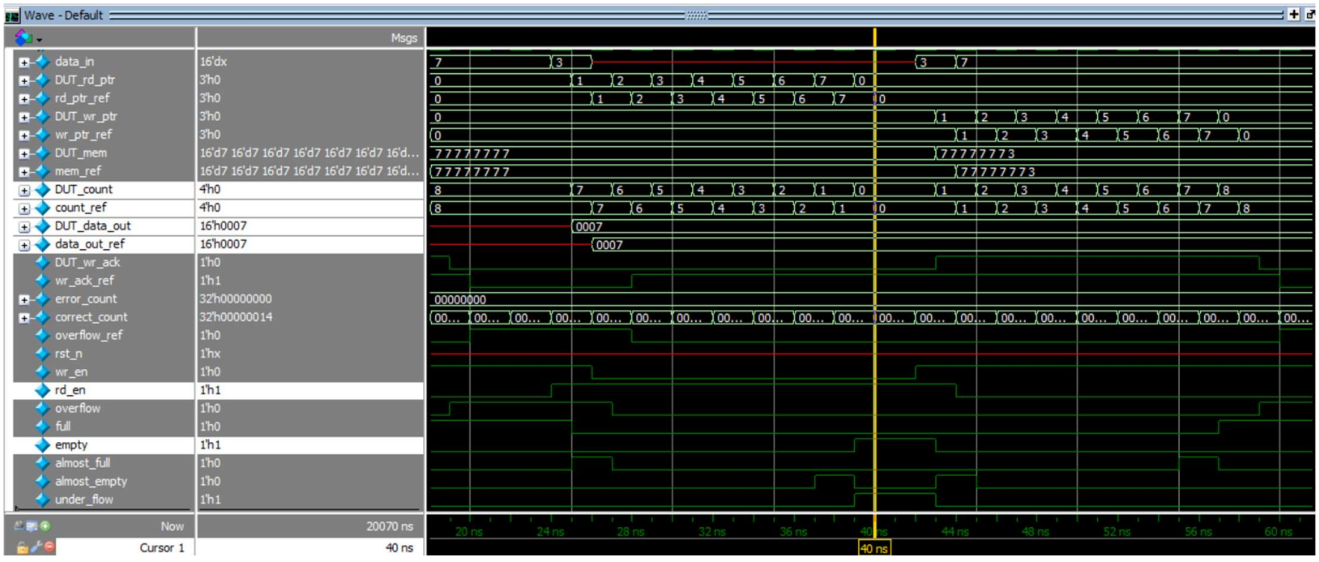
➢ Read Only Loop Sequence:

```
1. package FIFO_ReadOnly_Loop_Sequence_pkg;
2. import uvm_pkg::*;
3. import FIFO_Sequence_Item_pkg::*;
4. `include "uvm_macros.svh"
5.
6. class FIFO_ReadOnly_Loop_Sequence extends uvm_sequence #(FIFO_Sequence_Item);
7. `uvm_object_utils(FIFO_ReadOnly_Loop_Sequence)
8.
9. FIFO_Sequence_Item fifo_sequence_item;
10.
11.function new (string name = "FIFO_ReadOnly_Loop_Sequence");
12.     super.new(name);
13.endfunction
14.
15.task body();
16.
17.     repeat (7) begin
18.         fifo_sequence_item =
   FIFO_Sequence_Item::type_id::create("fifo_sequence_item");
19.         start_item(fifo_sequence_item);
20.
21.         fifo_sequence_item.rd_en = 1; //Force rd_en to 1
22.         fifo_sequence_item.wr_en = 0; //Force wr_en to 0
23.
24.         finish_item(fifo_sequence_item);
25.     end
26.endtask
27.endclass
28.endpackage
```



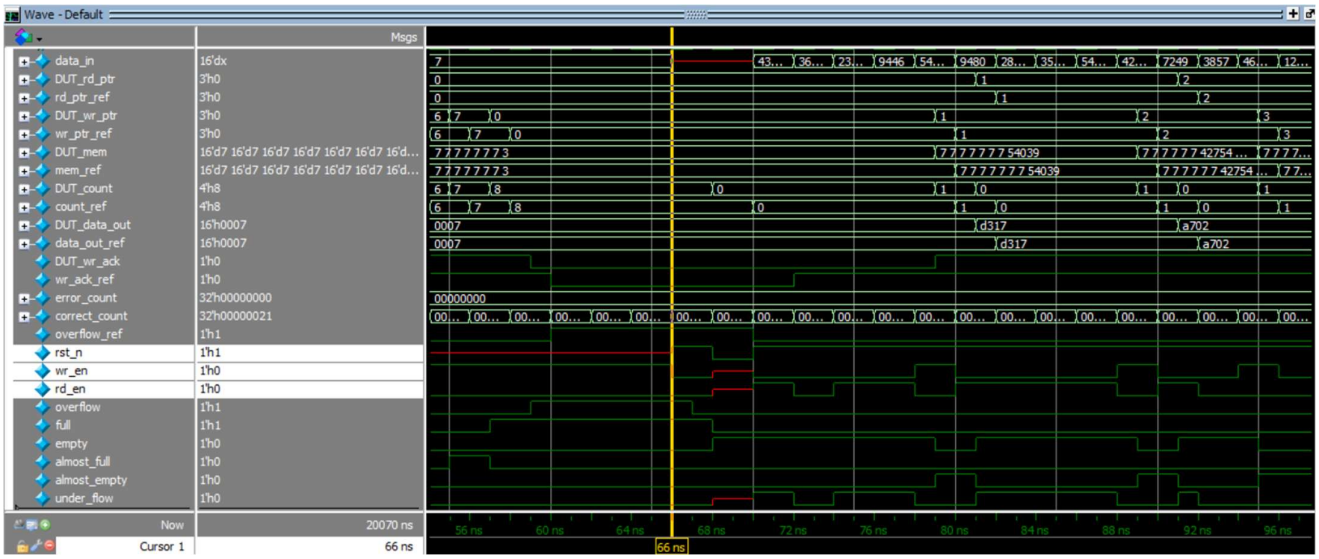
➢ Read Only Sequence:

```
1. package FIFO_ReadOnly_Sequence_pkg;
2. import uvm_pkg::*;
3. import FIFO_Sequence_Item_pkg::*;
4. `include "uvm_macros.svh"
5.
6. class FIFO_ReadOnly_Sequence extends uvm_sequence #(FIFO_Sequence_Item);
7. `uvm_object_utils(FIFO_ReadOnly_Sequence)
8.
9. FIFO_Sequence_Item fifo_sequence_item;
10.
11.function new (string name = "FIFO_ReadOnly_Sequence");
12.     super.new(name);
13.endfunction
14.
15.task body();
16.     fifo_sequence_item =
   FIFO_Sequence_Item::type_id::create("fifo_sequence_item");
17.     start_item(fifo_sequence_item);
18.     fifo_sequence_item.rd_en = 1; //Force rd_en to 1
19.     fifo_sequence_item.wr_en = 0; //Force wr_en to 0
20.     finish_item(fifo_sequence_item);
21.endtask
22.endclass
23.endpackage
```



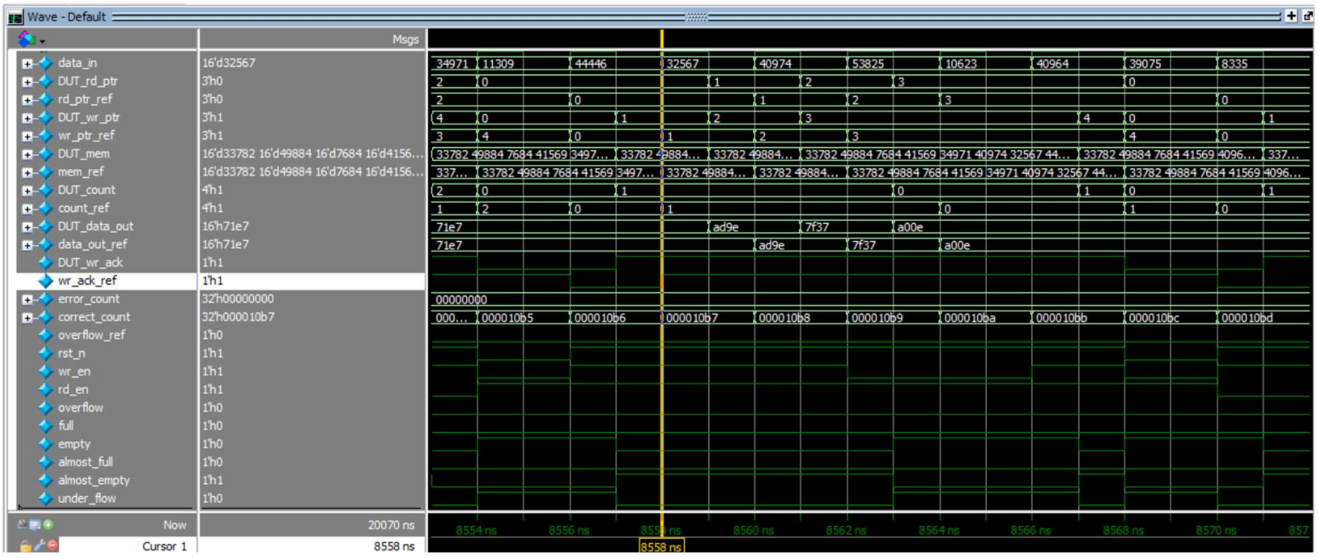
➢ Directed Sequence:

```
1. package FIFO_Directed_Sequence_pkg;
2. import uvm_pkg::*;
3. import FIFO_Sequence_Item_pkg::*;
4. `include "uvm_macros.svh"
5.
6. class FIFO_Directed_Sequence extends uvm_sequence #(FIFO_Sequence_Item);
7. `uvm_object_utils(FIFO_Directed_Sequence)
8.
9. FIFO_Sequence_Item fifo_sequence_item;
10.
11.function new (string name = "FIFO_Directed_Sequence");
12.     super.new(name);
13.endfunction
14.
15.task body();
16.
17.     fifo_sequence_item =
   FIFO_Sequence_Item::type_id::create("fifo_sequence_item");
18.     start_item(fifo_sequence_item);
19.
20.     //Needed to achieve 100% code coverage
21.     fifo_sequence_item.wr_en = 0; //Force wr_en to 0
22.
23.     fifo_sequence_item.rst_n = 1;
24.
25.     finish_item(fifo_sequence_item);
26.
27.endtask
28.
29.endclass
30.
31.endpackage
```



➢ Random Sequence:

```
1. package FIFO_Random_Sequence_pkg;
2. import uvm_pkg::*;
3. import FIFO_Sequence_Item_pkg::*;
4. `include "uvm_macros.svh"
5.
6. class FIFO_Random_Sequence extends uvm_sequence #(FIFO_Sequence_Item);
7. `uvm_object_utils(FIFO_Random_Sequence)
8.
9. FIFO_Sequence_Item fifo_sequence_item;
10.
11.function new (string name = "FIFO_Random_Sequence");
12.     super.new(name);
13.endfunction
14.
15.task body();
16.
17.repeat(10000) begin
18.     fifo_sequence_item =
   FIFO_Sequence_Item::type_id::create("fifo_sequence_item");
19.
20.     //Randomize the input variables.
21.     start_item(fifo_sequence_item);
22.     assert(fifo_sequence_item.randomize());
23.     finish_item(fifo_sequence_item);
24.
25.end
26.endtask
27.endclass
28.endpackage
```



➢ Sequence Item:

```
1. package FIFO_Sequence_Item_pkg;
2.
3. import uvm_pkg::*;
4. import FIFO_Shared_pkg::*;
5. `include "uvm_macros.svh"
6.
7. class FIFO_Sequence_Item extends uvm_sequence_item;
8. `uvm_object_utils(FIFO_Sequence_Item)
9.
10.rand logic [FIFO_WIDTH-1:0] data_in;
11.rand logic rst_n, wr_en, rd_en;
12.logic [FIFO_WIDTH-1:0] data_out;
13.logic wr_ack, overflow;
14.logic full, empty, almostfull, almostempty, underflow;
15.
16.integer RD_EN_ON_DIST=30,WR_EN_ON_DIST=70;
17.
18.//Constructor with default values for the rd_en and  Wr_en FIFO signals
19.function new(string name = "FIFO_Sequence_Item");
20.    super.new(name);
21.endfunction
22.
23.//Reset constraint
24.constraint rst_nCons{
25.    rst_n dist {0:=5,1:=95};
26.}
27.
28.//Constraint for write enable signal
29.constraint wr_enCons{
30.    wr_en dist {
31.        0:=WR_EN_ON_DIST, 1:=100-WR_EN_ON_DIST
32.    };
33.}
34.
35.//Constraint for read enable signal
36.constraint rd_enCons{
37.    rd_en dist {
38.        0:=RD_EN_ON_DIST, 1:=100-RD_EN_ON_DIST
39.    };
40.}
41.
42.endclass
43.
44.endpackage
```

➢ Top Module:

```
1.    import uvm_pkg::*;
2.    import FIFO_Test_pkg::*;
3.    `include "uvm_macros.svh"
4.
5.    module top();
6.
7.    bit clk;
8.      // Clock generation
9.    initial begin
10.       clk = 0;
11.       forever begin
12.       #1 clk = ~clk;
13.     end
14.   end
15.
16.   //Instantiate the interface and the DUT
17.   FIFO_IF fifo_if(clk);
18.   FIFO dut(fifo_if);
19.
20.   bind FIFO FIFO_Assertions fifo_inst (fifo_if);
21.
22.   initial begin
23.     uvm_config_db#(virtual FIFO_IF)::set(null, "uvm_test_top", "FIFO_IF", fifo_if);
      //Set the virtual interface in the configuration database
24.     run_test("FIFO_Test");
25.   end
26.
27.   endmodule
```

➢ Interface Code:

```
1.    import FIFO_Shared_pkg::*; // Import the shared package
2.
3.    interface FIFO_IF(clk);
4.
5.    input bit clk;
6.
7.    //Define the FIFO signals
8.    logic [FIFO_WIDTH-1:0] data_in;
9.    logic rst_n, wr_en, rd_en;
10.   logic [FIFO_WIDTH-1:0] data_out;
11.   logic wr_ack, overflow;
12.   logic full, empty, almostfull, almostempty, underflow;
13.
14.
15.   //Define modports for each of DUT, TB and Monitor respictively and drive the I/O
      signals
```

```
16.    modport DUT(input data_in, clk, rst_n, wr_en, rd_en, output data_out, wr_ack,
       overflow, full, empty, almostfull, almostempty, underflow);
17.    modport TEST(input data_out, wr_ack, overflow, full, empty, almostfull,
       almostempty, underflow, output data_in, clk, rst_n, wr_en, rd_en);
18.    modport MONITOR(input data_out, wr_ack, overflow, full, empty, almostfull,
       almostempty, underflow, clk, rst_n, data_in, wr_en, rd_en);
19.
20.    endinterface
```

➢ Test Code:

```
1.     package FIFO_Test_pkg;
2.     `include "uvm_macros.svh"
3.     import uvm_pkg::*;
4.     import FIFO_Random_Sequence_pkg::*;
5.     import FIFO_Reset_Sequence_pkg::*;
6.     import FIFO_Directed_Sequence_pkg::*;
7.     import FIFO_WriteOnly_Loop_Sequence_pkg::*;
8.     import FIFO_ReadOnly_Loop_Sequence_pkg::*;
9.     import FIFO_ReadWrite_Sequence_pkg::*;
10.    import FIFO_ReadOnly_Sequence_pkg::*;
11.    import FIFO_Env_pkg::*;
12.    import FIFO_Config_Obj_pkg::*;
13.
14.    class FIFO_Test extends uvm_test;
15.    `uvm_component_utils(FIFO_Test)
16.
17.    FIFO_Env fifo_env;
18.    FIFO_Config_Obj fifo_config_obj;
19.    FIFO_Random_Sequence fifo_random_sequence;
20.    FIFO_Directed_Sequence fifo_directed_sequence;
21.    FIFO_Reset_Sequence fifo_reset_sequence;
22.    FIFO_ReadOnly_Loop_Sequence fifo_readonly_loop_sequence;
23.    FIFO_WriteOnly_Loop_Sequence fifo_writeonly_loop_sequence;
24.    FIFO_ReadOnly_Sequence fifo_readonly_sequence;
25.    FIFO_ReadWrite_Sequence fifo_readwrite_sequence;
26.
27.    function new (string name = "FIFO_Test", uvm_component parent = null);
28.        super.new(name,parent);
29.    endfunction
30.
31.    function void build_phase(uvm_phase phase);
32.        super.build_phase(phase);
33.        fifo_env = FIFO_Env::type_id::create("fifo_env",this);
34.        fifo_config_obj = FIFO_Config_Obj::type_id::create("fifo_config_obj",this);
35.        fifo_random_sequence =
       FIFO_Random_Sequence::type_id::create("fifo_random_sequence",this);
36.        fifo_directed_sequence =
       FIFO_Directed_Sequence::type_id::create("fifo_directed_sequence",this);
37.        fifo_reset_sequence =
       FIFO_Reset_Sequence::type_id::create("fifo_reset_sequence",this);
```

```
38.         fifo_readonly_loop_sequence =
    FIFO_ReadOnly_Loop_Sequence::type_id::create("fifo_readonly_loop_sequence",this);
39.         fifo_writeonly_loop_sequence =
    FIFO_WriteOnly_Loop_Sequence::type_id::create("fifo_writeonly_loop_sequence",this
    );
40.         fifo_readonly_sequence =
    FIFO_ReadOnly_Sequence::type_id::create("fifo_readonly_sequence",this);
41.         fifo_readwrite_sequence =
    FIFO_ReadWrite_Sequence::type_id::create("fifo_readwrite_sequence",this);
42.
43.         if(!uvm_config_db #(virtual FIFO_IF) ::get(this, "",
    "FIFO_IF",fifo_config_obj.fifo_if)) begin
44.             `uvm_fatal("FIFO_IF","Virtual interface not found in the configuration
    database")
45.         end
46.         uvm_config_db#(FIFO_Config_Obj)::set(this,"*","fifo_config_obj",fifo_config_o
    bj); //Set the virtual interface in the configuration database
47.     endfunction
48.
49.     task run_phase(uvm_phase phase);
50.         super.run_phase(phase);
51.         phase.raise_objection(this);
52.
53.         `uvm_info("run phase","FIFO_1",UVM_LOW);
54.         fifo_reset_sequence.start(fifo_env.agent.sequencer);  // Start FIFO_1
55.         `uvm_info("run phase","FIFO_1 Finished",UVM_LOW);
56.
57.         `uvm_info("run phase","FIFO_2",UVM_LOW);
58.         fifo_writeonly_loop_sequence.start(fifo_env.agent.sequencer); // Start FIFO_2
59.         `uvm_info("run phase","FIFO_2 Finished",UVM_LOW);
60.
61.         `uvm_info("run phase","FIFO_3",UVM_LOW);
62.         fifo_readwrite_sequence.start(fifo_env.agent.sequencer); // Start FIFO_3
63.         `uvm_info("run phase","FIFO_3 Finished",UVM_LOW);
64.
65.         `uvm_info("run phase","FIFO_4",UVM_LOW);
66.         fifo_readonly_loop_sequence.start(fifo_env.agent.sequencer); // Start FIFO_4
67.         `uvm_info("run phase","FIFO_4 Finished",UVM_LOW);
68.
69.         `uvm_info("run phase","FIFO_5",UVM_LOW);
70.         fifo_readonly_sequence.start(fifo_env.agent.sequencer); // Start FIFO_5
71.         `uvm_info("run phase","FIFO_5 Finished",UVM_LOW);
72.
73.         `uvm_info("run phase","FIFO_6",UVM_LOW);
74.         fifo_readwrite_sequence.start(fifo_env.agent.sequencer); // Start FIFO_6
75.         `uvm_info("run phase","FIFO_6 Finished",UVM_LOW);
76.
77.         `uvm_info("run phase","FIFO_7",UVM_LOW);
78.         fifo_writeonly_loop_sequence.start(fifo_env.agent.sequencer); // Start FIFO_7
79.         `uvm_info("run phase","FIFO_7 Finished",UVM_LOW);
80.
81.         fifo_directed_sequence.start(fifo_env.agent.sequencer);
82.
83.         fifo_reset_sequence.start(fifo_env.agent.sequencer);
```

```
84.
85.         `uvm_info("run phase","FIFO_8",UVM_LOW);
86.         fifo_random_sequence.start(fifo_env.agent.sequencer); // Start FIFO_8
87.         `uvm_info("run phase","FIFO_8 Finished",UVM_LOW);
88.
89.         phase.drop_objection(this);
90.     endtask
91.
92.     endclass
93.     endpackage
```

➢ Environment Code:

```
1. package FIFO_Env_pkg;
2.
3. import uvm_pkg::*;
4. import FIFO_Agent_pkg::*;
5. import FIFO_Coverage_pkg::*;
6. import FIFO_Scoreboard_pkg::*;
7.
8. `include "uvm_macros.svh"
9.
10.class FIFO_Env extends uvm_env;
11.
12.`uvm_component_utils(FIFO_Env)
13.
14.FIFO_Agent agent; // Declare the agent
15.FIFO_Coverage coverage; //Declare the coverage
16.FIFO_Scoreboard scoreboard; //Declare the Scoreboard
17.
18.function new(string name = "FIFO_Env", uvm_component parent = null);
19.     super.new(name,parent);
20.endfunction
21.
22.function void build_phase(uvm_phase phase);
23.     super.build_phase(phase);
24.     agent = FIFO_Agent::type_id::create("agent",this); // Create the agent
25.     coverage = FIFO_Coverage::type_id::create("coverage",this); // Create the
   coverage object
26.     scoreboard = FIFO_Scoreboard::type_id::create("scoreboard",this); // Create the
   scoreboard
27.endfunction
28.
29.function void connect_phase(uvm_phase phase);
30.     agent.agt_ap.connect(scoreboard.sb_export); // Connect the agent's analysis port
   to the scoreboard
31.     agent.agt_ap.connect(coverage.cov_export); // Connect the agent's analysis port
   to the coverage object
32.endfunction
33.
34.endclass
35.
36.endpackage
```

➢ Scoreboard Code:

```
1. package FIFO_Scoreboard_pkg;
2.
3. import uvm_pkg::*;
4. import FIFO_Sequence_Item_pkg::*;
5. import FIFO_Shared_pkg::*;
6. `include "uvm_macros.svh"
7.
8. class FIFO_Scoreboard extends uvm_scoreboard;
9. `uvm_component_utils(FIFO_Scoreboard)
10.
11.logic [FIFO_WIDTH-1:0] data_out_ref;
12.logic wr_ack_ref,overflow_ref;
13.
14.//Generate the number of bits needed for accessing the entire FIFO memory
15.localparam max_fifo_addr = $clog2(FIFO_DEPTH);
16.
17.//FIFO memory array
18.logic [FIFO_WIDTH-1:0] mem_ref [FIFO_DEPTH-1:0];
19.
20.//Pointers to track the read and write positions in the FIFO
21.logic [max_fifo_addr-1:0] wr_ptr_ref, rd_ptr_ref;
22.
23.//Count to track the number of elements in the FIFO
24.logic [max_fifo_addr:0] count_ref;
25.
26.// Analysis port for the scoreboard
27.uvm_analysis_export #(FIFO_Sequence_Item) sb_export;
28.
29.// FIFO for the scoreboard
30.uvm_tlm_analysis_fifo #(FIFO_Sequence_Item) sb_fifo;
31.
32.int correct_count=0,error_count=0;
33.
34.FIFO_Sequence_Item fifo_sequence_item;
35.
36.function new (string name = "FIFO_Scoreboard", uvm_component parent = null);
37.     super.new(name,parent);
38.endfunction
39.
40.function void build_phase(uvm_phase phase);
41.
42.     super.build_phase(phase);
43.     sb_fifo = new("sb_fifo", this); // Create the FIFO
44.     sb_export = new("sb_export", this); // Create the analysis export
45.
46.endfunction
47.
48.function void connect_phase(uvm_phase phase);
49.
50.     super.connect_phase(phase);
```

```
51.     sb_export.connect(sb_fifo.analysis_export); // Connect the analysis export to
   the FIFO
52.
53.endfunction
54.
55.task run_phase(uvm_phase phase);
56.
57.     super.run_phase(phase);
58.     forever begin
59.         sb_fifo.get(fifo_sequence_item); // Get the sequence item from the FIFO
60.         `uvm_info("run_phase",fifo_sequence_item.convert2string(),UVM_MEDIUM); //
   Log the sequence item
61.
62.         ref_model(fifo_sequence_item); // Call the reference model to check the
   output
63.
64.         if(fifo_sequence_item.data_out != data_out_ref) begin
65.             `uvm_error("Scoreboard", $sformatf("Output mismatch detected! Expected:
   %0d, Got: %0d", data_out_ref, fifo_sequence_item.data_out));
66.             error_count++;
67.         end
68.         else correct_count++;
69.
70.     end
71.endtask
72.
73.task ref_model(FIFO_Sequence_Item fifo_sequence_item);
74.
75.     if(!fifo_sequence_item.rst_n) begin
76.             // Treat the FIFO as empty when reset is asserted
77.             wr_ptr_ref = 0;
78.             rd_ptr_ref = 0;
79.             count_ref = 0;
80.             overflow_ref = 0;
81.             wr_ack_ref = 0;
82.         end
83.
84.         //If both write and read are enabled :
85.         else if(fifo_sequence_item.wr_en && fifo_sequence_item.rd_en) begin
86.
87.             //If FIFO is empty, only write is allowed
88.             if(count_ref == 0) begin
89.                 mem_ref[wr_ptr_ref] = fifo_sequence_item.data_in;
90.                 wr_ack_ref = 1;
91.                 wr_ptr_ref++;
92.                 count_ref++;
93.             end
94.
95.             //If FIFO is full, only read is allowed
96.             else if(count_ref == FIFO_DEPTH) begin
97.                 data_out_ref = mem_ref[rd_ptr_ref];
98.                 rd_ptr_ref++;
99.                 count_ref--;
100.                  end
```

```
101.
102.                 else begin
103.                     mem_ref[wr_ptr_ref] = fifo_sequence_item.data_in;
104.                     wr_ack_ref = 1;
105.                     wr_ptr_ref++;
106.                     data_out_ref = mem_ref[rd_ptr_ref];
107.                     rd_ptr_ref++;
108.                 end
109.             end
110.             //If one of the write or read is only enabled:
111.             else if(fifo_sequence_item.wr_en || fifo_sequence_item.rd_en) begin
112.
113.                 //If only write is enabled:
114.                 if(fifo_sequence_item.wr_en && !fifo_sequence_item.rd_en &&
    count_ref < FIFO_DEPTH) begin
115.                     mem_ref[wr_ptr_ref] = fifo_sequence_item.data_in;
116.                     wr_ack_ref = 1;
117.                     wr_ptr_ref++;
118.                     count_ref++;
119.                 end
120.                 else begin //If write is not enabled or FIFO is full
121.
122.                     if ((count_ref == FIFO_DEPTH) && fifo_sequence_item.wr_en)
    begin
123.                         wr_ack_ref = 0;
124.                         overflow_ref = 1;
125.                     end else begin
126.                         wr_ack_ref = 1;
127.                         overflow_ref = 0;
128.                     end
129.                 end
130.
131.                 //If only read is enabled:
132.                 if(fifo_sequence_item.rd_en && !fifo_sequence_item.wr_en &&
    count_ref > 0) begin
133.                     data_out_ref = mem_ref[rd_ptr_ref];
134.                     rd_ptr_ref++;
135.                     count_ref--;
136.                 end
137.             end
138.
139.     endtask
140.
141.     function void report_phase(uvm_phase phase);
142.         super.report_phase(phase);
143.         `uvm_info("report_phase", $sformatf("Correct Count: %0d, Error Count:
    %0d", correct_count, error_count), UVM_MEDIUM);
144.     endfunction
145.
146.     endclass
147.
148.     endpackage
```

➢ Coverage Collector Code:

```
1.  package FIFO_Coverage_pkg;
2.
3.  import uvm_pkg::*;
4.  import FIFO_Sequence_Item_pkg::*;
5.  `include "uvm_macros.svh"
6.
7.  class FIFO_Coverage extends uvm_component;
8.  `uvm_component_utils(FIFO_Coverage)
9.  uvm_analysis_export #(FIFO_Sequence_Item) cov_export;
10.
11. uvm_tlm_analysis_fifo #(FIFO_Sequence_Item) cov_fifo;
12. FIFO_Sequence_Item fifo_sequence_item;
13.

14. covergroup cover_group;
15.
16.     //Create cover point for each control signal of the FIFO
17.     WRITE_ENABLE: coverpoint fifo_sequence_item.wr_en {}
18.     READ_ENABLE: coverpoint fifo_sequence_item.rd_en {}
19.     WRITE_ACK: coverpoint fifo_sequence_item.wr_ack {}
20.     OVERFLOW: coverpoint fifo_sequence_item.overflow {}
21.     UNDERFLOW: coverpoint fifo_sequence_item.underflow {}
22.     FULL: coverpoint fifo_sequence_item.full {}
23.     EMPTY: coverpoint fifo_sequence_item.empty {}
24.     ALMOSTFULL: coverpoint fifo_sequence_item.almostfull {}
25.     ALMOSTEMPTY: coverpoint fifo_sequence_item.almostempty {}
26.
27.
28.     //Cross covergroup between write, read and different output signals
29.     WRITE_READ_ACK_CROSS: cross WRITE_ENABLE, READ_ENABLE, WRITE_ACK {}
30.
31.     WRITE_READ_OVERFLOW_CROSS: cross WRITE_ENABLE, READ_ENABLE, OVERFLOW {
32.         ignore_bins bins_ignore1 = binsof(WRITE_ENABLE) intersect {0} &&
    binsof(READ_ENABLE) intersect {0}
33.         && binsof(OVERFLOW) intersect{1};
34.
35.         ignore_bins bins_ignore2 = binsof(WRITE_ENABLE) intersect {0} &&
    binsof(READ_ENABLE) intersect {1}
36.         && binsof(OVERFLOW) intersect{1};
37.     }
38.
39.     WRITE_READ_UNDERFLOW_CROSS: cross WRITE_ENABLE, READ_ENABLE, UNDERFLOW {
40.
41.         ignore_bins bins_ignore1 = binsof(WRITE_ENABLE) intersect {0} &&
    binsof(READ_ENABLE) intersect {0}
42.         && binsof(UNDERFLOW) intersect{1};
43.
44.         ignore_bins bins_ignore2 = binsof(WRITE_ENABLE) intersect {1} &&
    binsof(READ_ENABLE) intersect {0}
45.         && binsof(UNDERFLOW) intersect{1};
46.     }
```

```
47.
48.         WRITE_READ_FULL_CROSS: cross WRITE_ENABLE, READ_ENABLE, FULL {
49.
50.             ignore_bins bins_ignore1 = binsof(WRITE_ENABLE) intersect {0} &&
      binsof(READ_ENABLE) intersect {1}
51.             && binsof(FULL) intersect{1};
52.
53.             ignore_bins bins_ignore2 = binsof(WRITE_ENABLE) intersect {1} &&
      binsof(READ_ENABLE) intersect {1}
54.             && binsof(FULL) intersect{1};
55.         }
56.
57.         WRITE_READ_EMPTY_CROSS: cross WRITE_ENABLE, READ_ENABLE, EMPTY {}
58.
59.         WRITE_READ_ALMOSTFULL_CROSS: cross WRITE_ENABLE, READ_ENABLE, ALMOSTFULL {
60.             ignore_bins bins_ignore1 = binsof(WRITE_ENABLE) intersect {0} &&
      binsof(READ_ENABLE) intersect {0}
61.             && binsof(ALMOSTFULL) intersect{1};
62.
63.             ignore_bins bins_ignore2 = binsof(WRITE_ENABLE) intersect {0} &&
      binsof(READ_ENABLE) intersect {1}
64.             && binsof(ALMOSTFULL) intersect{1};
65.         }
66.
67.         WRITE_READ_ALMOSTEMPTY_CROSS: cross WRITE_ENABLE, READ_ENABLE, ALMOSTEMPTY {}
68.
69.     endgroup
70.
71.     function new (string name = "FIFO_Coverage", uvm_component parent = null);
72.         super.new(name,parent);
73.         cover_group = new();
74.     endfunction
75.
76.     function void build_phase(uvm_phase phase);
77.         super.build_phase(phase);
78.         cov_export = new("cov_export",this);
79.         cov_fifo = new("cov_fifo",this);
80.     endfunction
81.
82.     function void connect_phase(uvm_phase phase);
83.         super.connect_phase(phase);
84.         cov_export.connect(cov_fifo.analysis_export);
85.     endfunction
86.
87.     task run_phase(uvm_phase phase);
88.         super.run_phase(phase);
89.         forever begin
90.            cov_fifo.get(fifo_sequence_item); // Get the sequence item from the FIFO
91.            cover_group.sample();
92.         end
93.     endtask
94.
95.     endclass
96.     endpackage
```

➢ Agent Code:

```
1. package FIFO_Agent_pkg;
2. import uvm_pkg::*;
3. import FIFO_Driver_pkg::*;
4. import FIFO_Monitor_pkg::*;
5. import FIFO_Config_Obj_pkg::*;
6. import FIFO_Sequence_Item_pkg::*;
7. import FIFO_Sequencer_pkg::*;
8. `include "uvm_macros.svh"
9.
10.class FIFO_Agent extends uvm_agent;
11.`uvm_component_utils(FIFO_Agent)
12.
13.FIFO_Driver driver;
14.FIFO_Sequencer sequencer;
15.FIFO_Monitor monitor;
16.FIFO_Config_Obj fifo_config_obj;
17.uvm_analysis_port #(FIFO_Sequence_Item) agt_ap;
18.
19.function new (string name = "FIFO_Agent", uvm_component parent = null);
20.super.new(name,parent);
21.endfunction
22.
23.function void build_phase(uvm_phase phase);
24.super.build_phase(phase);
25.
26.if(!uvm_config_db #(FIFO_Config_Obj)::get(this,
   "","fifo_config_obj",fifo_config_obj)) begin
27.    `uvm_fatal("build_phase", "Agent- unable to get the config object");
28.end
29.
30.sequencer = FIFO_Sequencer::type_id::create("sequencer",this);
31.driver = FIFO_Driver::type_id::create("driver",this);
32.monitor = FIFO_Monitor::type_id::create("monitor",this);
33.
34.agt_ap = new ("agt_ap",this);
35.
36.endfunction
37.
38.function void connect_phase(uvm_phase phase);
39.
40.    super.connect_phase(phase);
41.
42.    //Connect the driver virtual interface to the interface obtained from the
   configuration object
43.    driver.fifo_driver_vif = fifo_config_obj.fifo_if;
44.
45.    //Connect the monitor virtual interface to the interface obtained from the
   configuration object
46.    monitor.fifo_if = fifo_config_obj.fifo_if;
47.
48.    driver.seq_item_port.connect(sequencer.seq_item_export);
```

```
    monitor.mon_ap.connect(agt_ap);

endfunction

endclass
endpackage
```

➢ Sequencer Code:

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

class FIFO_Sequencer extends uvm_sequencer #(FIFO_Sequence_Item);
   `uvm_component_utils(FIFO_Sequencer)

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

endclass

endpackage
```

➢ Driver Code:

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




class FIFO_Driver extends uvm_driver #(FIFO_Sequence_Item);
`uvm_component_utils(FIFO_Driver)

virtual FIFO_IF fifo_driver_vif; // Declare a virtual interface to be used in the
driver
FIFO_Sequence_Item fifo_sequence_item; // Declare a sequence item to be used in the
driver

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
```

```
        fifo_sequence_item = FIFO_Sequence_Item::type_id::create("fifo_sequence_item", this); // Create a sequence item
        seq_item_port.get_next_item(fifo_sequence_item); // Get the next item from the sequencer

        fifo_driver_vif.data_in = fifo_sequence_item.data_in;
        fifo_driver_vif.rst_n = fifo_sequence_item.rst_n;
        fifo_driver_vif.wr_en = fifo_sequence_item.wr_en;
        fifo_driver_vif.rd_en = fifo_sequence_item.rd_en;

        @(negedge fifo_driver_vif.clk);
        seq_item_port.item_done(); // Indicate that the item is done
        `uvm_info("run_phase",fifo_sequence_item.convert2string(),UVM_HIGH); // Display the sequence item values
    end

endtask

endclass
endpackage
```

➢ Monitor Code:

```
package FIFO_Monitor_pkg;

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

class FIFO_Monitor extends uvm_monitor;
    `uvm_component_utils(FIFO_Monitor)
    virtual FIFO_IF fifo_if;
    FIFO_Sequence_Item fifo_sequence_item;
    uvm_analysis_port #(FIFO_Sequence_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
        fifo_sequence_item = FIFO_Sequence_Item::type_id::create("fifo_sequence_item");
```

```
            @(negedge fifo_if.clk);
            fifo_sequence_item.data_in = fifo_if.data_in;
            fifo_sequence_item.rst_n = fifo_if.rst_n;
            fifo_sequence_item.wr_en = fifo_if.wr_en;
            fifo_sequence_item.rd_en = fifo_if.rd_en;
            fifo_sequence_item.data_out = fifo_if.data_out;
            fifo_sequence_item.wr_ack = fifo_if.wr_ack;
            fifo_sequence_item.full = fifo_if.full;
            fifo_sequence_item.empty = fifo_if.empty;
            fifo_sequence_item.almostfull = fifo_if.almostfull;
            fifo_sequence_item.almostempty = fifo_if.almostempty;
            fifo_sequence_item.underflow = fifo_if.underflow;
            fifo_sequence_item.overflow = fifo_if.overflow;
            mon_ap.write(fifo_sequence_item);
            `uvm_info("run_phase",fifo_sequence_item.convert2string(),UVM_MEDIUM);
        end
    endtask

    endclass

    endpackage
```

➢ 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_if;

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

    endclass
    endpackage
```

➢ Shared Package:

```
    package FIFO_Shared_pkg;

    //Define the FIFO_WIDTH and FIFO_DEPTH parameters
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;

    //Define Error and correct count variables
    integer error_count, correct_count;

    endpackage

```