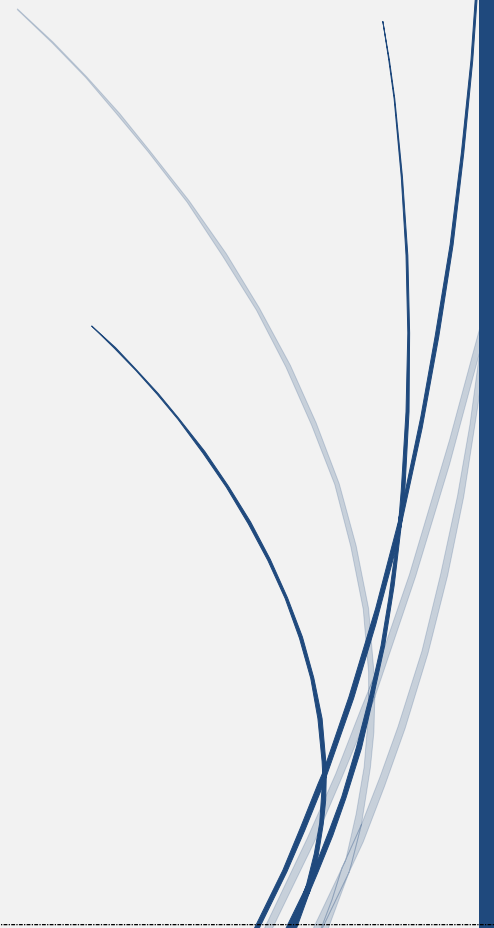


# UVM-based Verification

Synchronous FIFO

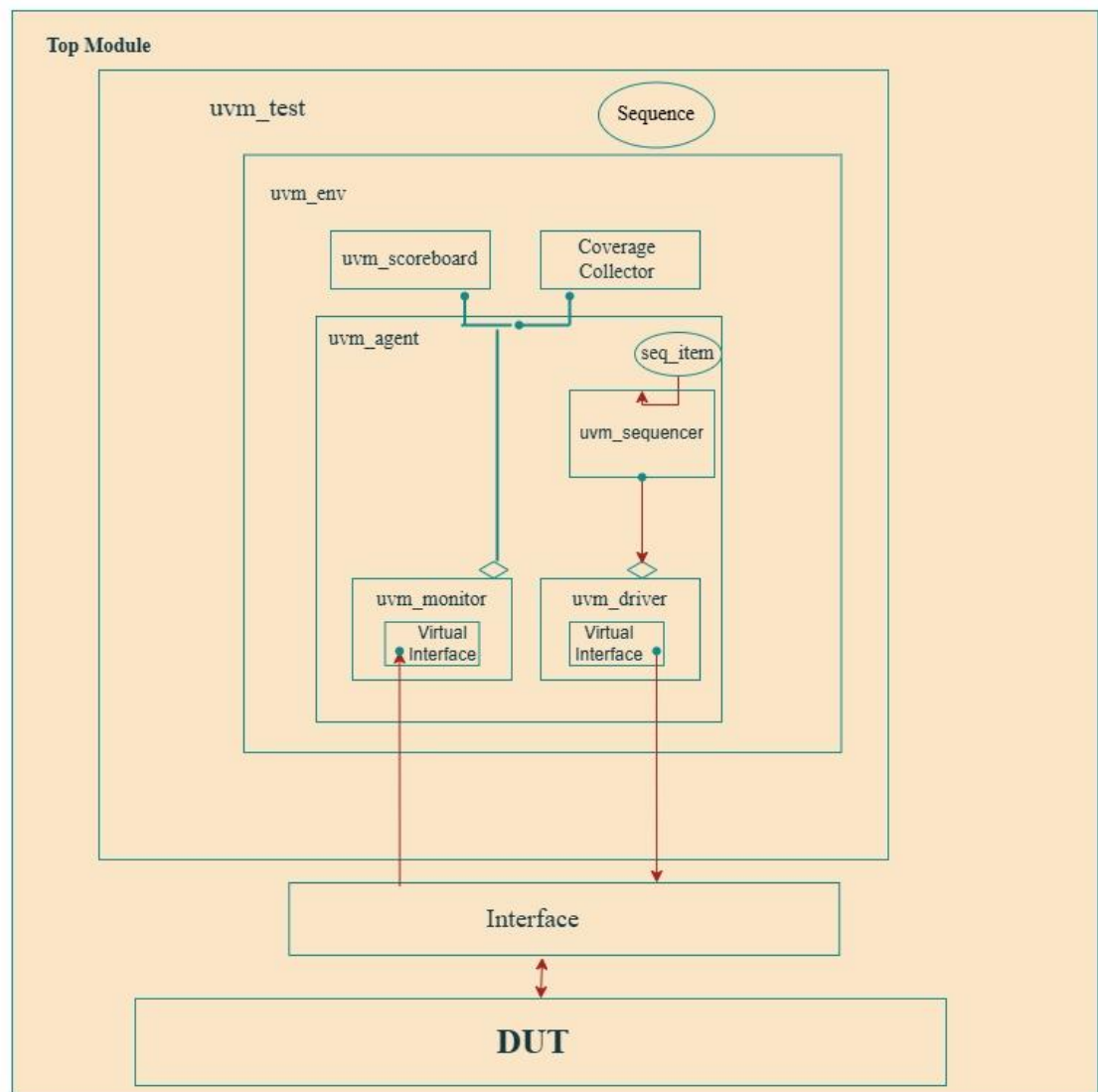
By Moustafa Mohammed



# 1- Verification Plan:

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	When reset is active FIFO should be empty , empty should be high the and internal counters(wr_ptr/rd_ptr/count) should be low	Directed at the start of simulation then randomized to be inactive most of time	-	<b>reset</b> :Immediate Assertion to check the functionality of asynchronous reset.
FIFO_2	When write enable is asserted while the FIFO is not full the data_in should be loaded to FIFO and wr_ptr should increment	Randomization during the simulation with probability related to WR_EN_ON_DIST value	-	<b>wr_ptr_a</b> :Concurrent Assertion to check the functionality of write enable and wr_ptr.
FIFO_3	When read enable is asserted while the FIFO is not empty the data_out should be loaded from FIFO and rd_ptr should increment	Randomization during the simulation with probability related to RD_EN_ON_DIST value	-	<b>rd_ptr_a</b> :Concurrent Assertion to check the functionality of read enable.
FIFO_4	When write enable is asserted while the FIFO is not full the wr_ack signal is asserted	Randomization during the simulation with probability related to WR_EN_ON_DIST value	-	<b>wr_ack_a</b> :Concurrent Assertion to check the wr_ack flag.
FIFO_5	When write enable is asserted while the FIFO is full, overflow signal is asserted	Randomization during the simulation with probability related to WR_EN_ON_DIST value	-	<b>overflow_a</b> :Concurrent Assertion to check the overflow flag.
FIFO_6	When read enable is asserted while the FIFO is empty ,underflow signal is asserted	Randomization during the simulation with probability related to RD_EN_ON_DIST value	-	<b>underflow_a</b> :Concurrent Assertion to check the underflow flag.
FIFO_7	When write enable is asserted and read enable is deasserted and FIFO is not full the internal signal count should increment	Randomization during the simulation with probability related to WR_EN_ON_DIST value	-	<b>count_inc_a</b> :Concurrent Assertion to check the count signal increment correct.
FIFO_8	When read enable is Deasserted and write enable is asserted while the FIFO is not empty the internal signal count should decrement	Randomization during the simulation with probability related to RD_EN_ON_DIST value	-	<b>count_dec_a</b> :Concurrent Assertion to check the count signal decrement correct.
FIFO_9	When read enable is asserted and write enable is asserted while the FIFO is neither full nor empty	Randomization during the simulation with probability related to RD_EN_ON_DIST and WR_EN_ON_DIST values	-	<b>count_no_chang_a</b> :Concurrent Assertion to check the count signal does not change.
FIFO_10	When Count equals to zero means the FIFO is empty	-	-	<b>empty_check</b> :Immediate Assertion to check the empty flag.
FIFO_11	When Count equals to one means the FIFO is written once so its almostempty	-	-	<b>almostempty_check</b> :Immediate Assertion to check the almostempty flag.
FIFO_12	When Count equals to FIFO_DEPTH means the FIFO is written FIFO_DEPTH times so its full	-	-	<b>full_check</b> :Immediate Assertion to check the empty flag.
FIFO_13	When Count equals to FIFO_DEPTH-1 means the FIFO is written FIFO_DEPTH-1 times so it's almostfull	-	-	<b>almost_full_check</b> :Immediate Assertion to check the empty flag.
FIFO_14	When read enable is asserted while the FIFO is not empty the data_out should be loaded from FIFO	-	-	A Checker in the Scoreboard checks for the correct functionality of write to and read from FIFO
FIFO_15	Combinations of wr_en,rd_en ,and almostempty flag	-	cross cover values of write,read enables and almostempty.	-
FIFO_16	Combinations of wr_en,rd_en ,and empty flag	-	cross cover values of write,read enables and empty.	-
FIFO_17	Combinations of wr_en,rd_en ,and almostfull flag	-	cross cover values of write,read enables and almostfull.	-
FIFO_15	Combinations of wr_en,rd_en ,and almostempty flag	-	cross cover values of write,read enables and almostempty.	-
FIFO_16	Combinations of wr_en,rd_en ,and empty flag	-	cross cover values of write,read enables and empty.	-
FIFO_17	Combinations of wr_en,rd_en ,and almostfull flag	-	cross cover values of write,read enables and almostfull.	-
FIFO_18	Combinations of wr_en,rd_en ,and full flag	-	cross cover values of write,read enables and full.	-
FIFO_19	Combinations of wr_en,rd_en ,and overflow flag	-	cross cover values of write,read enables and overflow.	-
FIFO_20	Combinations of wr_en,rd_en ,and underflow flag	-	cross cover values of write,read enables and underflow.	-
FIFO_21	Combinations of wr_en,rd_en ,and wr_ack flag	-	cross cover values of write,read enables and wr_ack.	-

## 2- UVM Testbench Structure:



### UVM testbench flow:

1. Top Module generates the clock, instantiates DUT Interface, DUT, and binds assertions module (which checks for FIFO flags and internal signals) then it passes virtual interface pointing to the DUT interface to Configuration database and call global task run\_test to create and run the testbench.
2. Test build the environment component and the sequences (Reset Sequence – Write Sequence – Read Sequence – Write Read Sequence ). Then test retrieves the virtual interface to the virtual interface of Configuration

object and passes the configuration object to configuration database then it starts the sequences on the sequencer.

3. Environment Builds the Agent, coverage collector, and scoreboard ( Reference Model ) then connects the analysis port of the agent with scoreboard and coverage collector analysis ports.

4. Agent builds Driver , Sequencer , and Monitor Components and connects the sequencer to the driver and it retrieves the configuration object interface and assign it to monitor and driver interfaces and connects the monitor analysis port with agent analysis port.

5. When Test starts sequence on the sequencer the sequence items sent to sequencer and the sequencer passes it to the driver when it requests the sequence item.

6. Driver Component requests sequence item from sequencer it drives it to the interface.

7. Monitor Component monitors the DUT Interface activity and translates it to transactions then it broadcasts them to the analysis component (coverage collector – scoreboard).

8. Coverage Collector get the sequence items from monitor then it samples the data for functional coverage.

9. Scoreboard gets the sequence items from monitor then checks correct functionality of data\_out by reference model which calculates the data\_out reference and compares it with the DUT data\_out.

### 3- RTL:

```
8  module FIFO(FIFO_Interface.DUT FIFO_if);
9
10  localparam max_fifo_addr = $clog2(FIFO_if.FIFO_DEPTH);
11
12  reg [FIFO_if.FIFO_WIDTH-1:0] mem [FIFO_if.FIFO_DEPTH-1:0];
13
14  reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
15  reg [max_fifo_addr:0] count;
16
17  always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
18      if (!FIFO_if.rst_n) begin
19          wr_ptr <= 0;
20          FIFO_if.wr_ack <= 0;
21          FIFO_if.overflow <= 0;
22      end
23      else if (FIFO_if.wr_en && count < FIFO_if.FIFO_DEPTH) begin
24          mem[wr_ptr] <= FIFO_if.data_in;
25          FIFO_if.wr_ack <= 1;
26          wr_ptr <= wr_ptr + 1;
27      end
28      else begin
29          FIFO_if.wr_ack <= 0;
30          if (FIFO_if.full && FIFO_if.wr_en)
31              FIFO_if.overflow <= 1;
32          else
```

```

33         FIFO_if.overflow <= 0;
34     end
35 end
36
37 always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
38     if (!FIFO_if.rst_n) begin
39         rd_ptr <= 0;
40         FIFO_if.underflow <= 0;
41     end
42     else if (FIFO_if.rd_en && count != 0) begin
43         FIFO_if.data_out <= mem[rd_ptr];
44         rd_ptr <= rd_ptr + 1;
45     end
46     else begin
47         if(FIFO_if.empty && FIFO_if.rd_en )
48             FIFO_if.underflow <= 1;
49         else
50             FIFO_if.underflow <= 0;
51     end
52 end
53
54 always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
55     if (!FIFO_if.rst_n) begin
56         count <= 0;
57     end
58     else begin
59         if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b10) && !FIFO_if.full)
60             count <= count + 1;
61         else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b01) && !FIFO_if.empty)
62             count <= count + 1;
63         else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.empty) begin
64             count <= count + 1;
65         end
66         else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.full)
67             count <= count - 1;
68     end
69 end
70
71 assign FIFO_if.full = (count == FIFO_if.FIFO_DEPTH)? 1 : 0;
72 assign FIFO_if.empty = (count == 0)? 1 : 0;
73 //assign FIFO_if.underflow = (FIFO_if.empty && FIFO_if.rd_en)? 1 : 0; // this is seq output not comb
74 assign FIFO_if.almostfull = (count == FIFO_if.FIFO_DEPTH-1)? 1 : 0; // should be FIFO_if.FIFO_DEPTH-1 not FIFO_if.FIFO_DEPTH-2
75 assign FIFO_if.almostempty = (count == 1)? 1 : 0;
76
77 endmodule

```



## 4- Assertions:

```
1  module FIFO_assertions(FIFO_Interface.DUT FIFO_if);
2
3  property wr_ack_p;
4  |@ (posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && !FIFO_if.full && !FIFO_if.rd_en) | =>
5  |FIFO_if.wr_ack ;
6  endproperty
7
8  wr_ack_a:assert property (wr_ack_p);
9  wr_ack_c:cover property (wr_ack_p);
10
11 property overflow_p;
12 |@ (posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && FIFO_if.full) | => FIFO_if.overflow;
13 endproperty
14
15 overflow_a:assert property (overflow_p);
16 overflow_c:cover property (overflow_p);
17
18 property underflow_p;
19 |@ (posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.rd_en && FIFO_if.empty) | => FIFO_if.underflow;
20 endproperty
21
22 underflow_a:assert property (underflow_p);
23 underflow_c:cover property (underflow_p);
24
25
26 property wr_ptr_p;
27 |@ (posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && !FIFO_if.full) | =>
28 |FIFO_DUT.wr_ptr == $past(FIFO_DUT.wr_ptr)+1'b1;
29 endproperty
30
31 wr_ptr_a:assert property(wr_ptr_p);
32 wr_ptr_c:cover property(wr_ptr_p);
33
34
35 property rd_ptr_p;
36 |@ (posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.rd_en && !FIFO_if.empty) | =>
37 |FIFO_DUT.rd_ptr == $past(FIFO_DUT.rd_ptr)+1'b1;
38 endproperty
39
40 rd_ptr_a:assert property(rd_ptr_p);
41 rd_ptr_c:cover property(rd_ptr_p);
42
43
44 property count_inc_p;
45 |@ (posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && !FIFO_if.rd_en && !FIFO_if.full) | =>
46 |FIFO_DUT.count == $past(FIFO_DUT.count)+1'b1;
47 endproperty
48
49 count_inc_a:assert property(count_inc_p);
50 count_inc_c:cover property(count_inc_p);
51
52 property count_dec_p;
53 |@ (posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (!FIFO_if.wr_en && FIFO_if.rd_en && !FIFO_if.empty) | =>
54 |FIFO_DUT.count == $past(FIFO_DUT.count)-1'b1;
55 endproperty
56
57 count_dec_a:assert property(count_dec_p);
58 count_dec_c:cover property(count_dec_p);
59
60
61 property count_no_change_p;
62 |@ (posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && FIFO_if.rd_en && !FIFO_if.full && !FIFO_if.empty ) | =>
63 |FIFO_DUT.count == $past(FIFO_DUT.count);
64 endproperty
65
```

```

65
66 count_no_change_a:assert property(count_no_change_p);
67 count_no_change_c:cover property(count_no_change_p);
68
69
70
71
72 always_comb begin : comb_outputs
73     if(FIFO_if.rst_n) begin
74
75         if(FIFO_DUT.count == FIFO_if.FIFO_DEPTH) begin
76             full_check:assert(FIFO_if.full == 1'b1);
77         end
78
79         if(FIFO_DUT.count == FIFO_if.FIFO_DEPTH-1) begin
80             almost_full_check:assert(FIFO_if.almostfull == 1'b1);
81         end
82
83         if(FIFO_DUT.count == 0) begin
84             empty_check:assert(FIFO_if.empty == 1'b1);
85         end
86
87         if(FIFO_DUT.count == 1) begin
88             almostempty_check:assert(FIFO_if.almostempty == 1'b1);
89         end
90
91     end
92 end
93
94 end
95

```

```

95
96 always_comb begin :reset_outputs
97     if(!FIFO_if.rst_n)
98         reset:assert final (FIFO_DUT.count == 0 && FIFO_DUT.wr_ptr == 0 && FIFO_DUT.rd_ptr == 0 && FIFO_if.empty == 1'b1 && FIFO_if.full == 0
99                             && FIFO_if.almostempty == 0 && FIFO_if.almostfull == 0 && FIFO_if.wr_ack == 0 );
100 end
101
102
103 endmodule

```

## 5- Assertions Table:

Feature	Assertion
When reset is active FIFO should be empty , empty should be high the and internal counters(wr_ptr/rd_ptr/count) should be low	assert final (FIFO_DUT.count == 0 && FIFO_DUT.wr_ptr == 0 && FIFO_DUT.rd_ptr == 0 && FIFO_if.empty == 1'b1 && FIFO_if.full == 0 && FIFO_if.almostempty == 0 && FIFO_if.almostfull == 0 && FIFO_if.wr_ack == 0 )
When write enable is asserted while the FIFO is not full the data_in should be loaded to FIFO and wr_ptr should increment	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && !FIFO_if.full)  >= (FIFO_DUT.wr_ptr == \$past(FIFO_DUT.wr_ptr)+1'b1);
When read enable is asserted while the FIFO is not empty the data_out should be loaded from FIFO and rd_ptr should increment	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.rd_en && !FIFO_if.empty)  >= (FIFO_DUT.rd_ptr == \$past(FIFO_DUT.rd_ptr)+1'b1);
When write enable is asserted while the FIFO is not full the wr_ack signal is asserted	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && !FIFO_if.full && !FIFO_if.rd_en)  >= FIFO_if.wr_ack ;
When write enable is asserted while the FIFO is full, overflow signal is asserted	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && FIFO_if.full)  >= FIFO_if.overflow
When read enable is asserted while the FIFO is empty ,underflow signal is asserted	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.rd_en && FIFO_if.empty)  >= FIFO_if.underflow



When write enable is asserted and read enable is deasserted and FIFO is not full the internal signal count should increment	<pre> @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en &amp;&amp; !FIFO_if.rd_en &amp;&amp; !FIFO_if.full)   =&gt; (FIFO_DUT.count == \$past(FIFO_DUT.count)+1'b1) </pre>
When read enable is Deasserted and write enable is asserted while the FIFO is not empty the internal signal count should decrement	<pre> @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (!FIFO_if.wr_en &amp;&amp; FIFO_if.rd_en &amp;&amp; !FIFO_if.empty)   =&gt; (FIFO_DUT.count == \$past(FIFO_DUT.count)-1'b1) </pre>
When read enable is asserted and write enable is asserted while the FIFO is neither full nor empty	<pre> @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en &amp;&amp; FIFO_if.rd_en &amp;&amp; !FIFO_if.full &amp;&amp; !FIFO_if.empty )   =&gt; (FIFO_DUT.count == \$past(FIFO_DUT.count)) </pre>
When Count equals to zero means the FIFO is empty	<pre> if(FIFO_DUT.count == 0) begin     empty_check:assert(FIFO_if.empty == 1'b1); end </pre>
When Count equals to one means the FIFO is written once so its almostempty	<pre> if(FIFO_DUT.count == 1) begin     almostempty_check:assert(FIFO_if.almostempty == 1'b1); end </pre>
When Count equals to FIFO_DEPTH means the FIFO is written FIFO_DEPTH times so its full	<pre> if(FIFO_DUT.count == FIFO_if.FIFO_DEPTH) begin     full_check:assert(FIFO_if.full == 1'b1); end </pre>
When Count equals to FIFO_DEPTH means the FIFO is written FIFO_DEPTH times so its full	<pre> if(FIFO_DUT.count == FIFO_if.FIFO_DEPTH) begin     full_check:assert(FIFO_if.full == 1'b1); end </pre>
When Count equals to FIFO_DEPTH-1 means the FIFO is written FIFO_DEPTH-1 times so it's almostfull	<pre> if(FIFO_DUT.count == FIFO_if.FIFO_DEPTH-1) begin     almost_full_check:assert(FIFO_if.almostfull == 1'b1); end </pre>

## 6- Interface :

```

1  interface FIFO_Interface(clk);
2
3  // FIFO Parameters //
4  parameter FIFO_WIDTH = 16;
5  parameter FIFO_DEPTH = 8 ;
6
7  input clk;
8
9  // FIFO Signals //
10 bit rst_n;
11 bit [FIFO_WIDTH-1 :0] data_in;
12 bit wr_en,rd_en;
13 bit [FIFO_WIDTH-1 :0] data_out;
14 logic almostfull,full;
15 logic almostempty,empty;
16 logic overflow,underflow;
17 logic wr_ack;
18
19 modport DUT (input clk, rst_n, wr_en, rd_en, data_in , output data_out, almostfull, full,
20             almostempty, empty, overflow, underflow, wr_ack );
21 endinterface

```

## 7- Verification Environment:

### 1.Top Module:

```
1  import uvm_pkg::*;
2  `include "uvm_macros.svh"
3  import FIFO_test_pkg::*;
4  module FIFO_top();
5
6  bit clk ;
7
8  always begin
9      clk = 1'b1;
10     #1;
11     clk = 1'b0;
12     #1;
13 end
14
15 FIFO_Interface FIFO_if(clk);
16
17 FIFO FIFO_DUT (FIFO_if);
18
19 bind FIFO FIFO_assertions FIFO_SVA (FIFO_if);
20
21 initial begin
22     uvm_config_db#(virtual FIFO_Interface)::set(null, "*", "FIFO_IF", FIFO_if);
23     run_test("FIFO_test");
24 end
25
26
27
28 endmodule
```

### 2. Configuration Object:

```
1  package FIFO_config_obj_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4
5  class FIFO_config_obj extends uvm_object ;
6  `uvm_object_utils(FIFO_config_obj)
7
8  virtual FIFO_Interface FIFO_vif;
9
10 function new ( string name = "FIFO_config_obj");
11
12     super.new(name);
13
14 endfunction
15
16 endclass
17
18 endpackage
```

### 3. Test:

```
1  package FIFO_test_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4  import FIFO_sequence_item_pkg::*;
5  import FIFO_sequence_pkg::*;
6  import FIFO_env_pkg::*;
7  import FIFO_config_obj_pkg::*;
8
9  class FIFO_test extends uvm_test;
10 `uvm_component_utils(FIFO_test)
11
12  FIFO_config_obj FIFO_cfg;
13  FIFO_env env;
14  FIFO_reset_sequence rst_seq;
15  FIFO_write_sequence wr_seq;
16  FIFO_read_sequence rd_seq;
17  FIFO_write_read_sequence wr_rd_seq;
18
19  function new (string name = "FIFO_test" , 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  env = FIFO_env::type_id::create("env",this);
26  FIFO_cfg = FIFO_config_obj::type_id::create("FIFO_cfg");
27  wr_seq = FIFO_write_sequence::type_id::create("wr_seq");
28  rd_seq = FIFO_read_sequence::type_id::create("rd_seq");
29  wr_rd_seq = FIFO_write_read_sequence::type_id::create("wr_rd_seq");
30  rst_seq = FIFO_reset_sequence::type_id::create("rst_seq");
31
```

```

32 if(! uvm_config_db #(virtual FIFO_Interface)::get(this,"", "FIFO_IF",FIFO_cfg.FIFO_vif)) begin
33 |   `uvm_fatal("build_phase","Test failed to get the virtual interface from configuration database");
34 end
35
36 uvm_config_db #(FIFO_config_obj)::set(this , "*" , "cfg" ,FIFO_cfg);
37
38 endfunction
39
40 task run_phase(uvm_phase phase);
41 super.run_phase(phase);
42 phase.raise_objection(this);
43
44 // Reset Sequence //
45 `uvm_info("run_phase","Reset Asserted",UVM_LOW);
46 rst_seq.start(env.agent.sqr);
47 `uvm_info("run_phase","Reset Deasserted",UVM_LOW);
48
49 // Write Read Sequence //
50 `uvm_info("run_phase","Write Read Sequence has Started",UVM_LOW);
51 wr_rd_seq.start(env.agent.sqr);
52 `uvm_info("run_phase","Write Read Sequence has Ended",UVM_LOW);
53
54 // Write only Sequence //
55 `uvm_info("run_phase","Write only Sequence has Started",UVM_LOW);
56 wr_seq.start(env.agent.sqr);
57 `uvm_info("run_phase","Write only Sequence has Ended",UVM_LOW);
58
59 // Read only Sequence //
60 `uvm_info("run_phase","Read only Sequence has Started",UVM_LOW);
61 rd_seq.start(env.agent.sqr);
62 `uvm_info("run_phase","Read only Sequence has Ended",UVM_LOW);
63
64
65
66
67
68
69
70
71
72
73

```

```

54 // Write only Sequence //
55 `uvm_info("run_phase","Write only Sequence has Started",UVM_LOW);
56 wr_seq.start(env.agent.sqr);
57 `uvm_info("run_phase","Write only Sequence has Ended",UVM_LOW);
58
59 // Read only Sequence //
60 `uvm_info("run_phase","Read only Sequence has Started",UVM_LOW);
61 rd_seq.start(env.agent.sqr);
62 `uvm_info("run_phase","Read only Sequence has Ended",UVM_LOW);
63
64 phase.drop_objection(this);
65
66
67 endtask
68
69
70
71 endclass
72
73 endpackage

```



## 4. Environment:

```
1  package FIFO_env_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4  import FIFO_scoreboard_pkg::*;
5  import FIFO_agent_pkg::*;
6  import FIFO_coverage_pkg::*;
7
8  class FIFO_env extends uvm_env;
9  `uvm_component_utils(FIFO_env)
10
11  FIFO_agent agent;
12  FIFO_coverage cov;
13  FIFO_scoreboard sb;
14
15  function new ( string name = "FIFO_env" , uvm_component parent = null );
16
17  super.new(name,parent);
18
19  endfunction
20
21  function void build_phase (uvm_phase phase);
22  |   super.build_phase(phase);
23  |   cov = FIFO_coverage::type_id::create("cov",this);
24  |   sb  = FIFO_scoreboard::type_id::create("sb",this);
25  |   agent = FIFO_agent::type_id::create("agent",this);
26  endfunction
27
28  function void connect_phase (uvm_phase phase);
29  |   super.connect_phase(phase);
30  |   agent.agent_ap.connect(sb.scoreboard_export);
31  |   agent.agent_ap.connect(cov.cov_export);
32
33  endfunction
34
35
36  endclass
37
38
39  endpackage
```

## 5- Agent:

```
1  package FIFO_agent_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4  import FIFO_sequence_item_pkg::*;
5  import FIFO_config_obj_pkg::*;
6  import FIFO_driver_pkg::*;
7  import FIFO_sequencer_pkg::*;
8  import FIFO_monitor_pkg::*;
9  class FIFO_agent extends uvm_agent;
10 `uvm_component_utils(FIFO_agent)
11 FIFO_driver drv;
12 FIFO_monitor mon;
13 FIFO_sequencer sqr;
14 FIFO_config_obj FIFO_cfg;
15 uvm_analysis_port #(FIFO_sequence_item) agent_ap;
16
17 function new ( string name = "FIFO_agent" , uvm_component parent = null );
18
19 super.new(name,parent);
20
21 endfunction
22
23 function void build_phase (uvm_phase phase);
24     super.build_phase(phase);
25     drv = FIFO_driver::type_id::create("drv",this);
26     mon = FIFO_monitor::type_id::create("mon",this);
27     sqr = FIFO_sequencer::type_id::create("sqr",this);
28     agent_ap = new ("agent_ap",this);
29     if(!uvm_config_db #(FIFO_config_obj)::get(this,"", "cfg",FIFO_cfg)) begin
30         `uvm_fatal("build_phase","Agent unable to get the configuration object from config database");
31     end
32 endfunction
33
34 function void connect_phase (uvm_phase phase);
35     super.connect_phase(phase);
36     drv.FIFO_vif = FIFO_cfg.FIFO_vif;
37     mon.FIFO_vif = FIFO_cfg.FIFO_vif;
38     drv.seq_item_port.connect(sqr.seq_item_export);
39     mon.mon_ap.connect(agent_ap);
40 endfunction
41
42
43 endclass
44 endpackage
```

## 6. Scoreboard:

```
1  package FIFO_scoreboard_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4  import FIFO_sequence_item_pkg::*;
5  import FIFO_config_obj_pkg::*;
6  import shared_pkg::*;
7  class FIFO_scoreboard extends uvm_scoreboard;
8  `uvm_component_utils(FIFO_scoreboard)
9
10     FIFO_sequence_item scoreboard_seq_item;
11
12     bit [FIFO_WIDTH-1:0] mem [$];
13     bit [FIFO_WIDTH-1:0] data_out_ref;
14     int correct_count=0;
15     int error_count = 0;
16     int count=0;
17
18     uvm_analysis_export #(FIFO_sequence_item) scoreboard_export;
19     uvm_tlm_analysis_fifo #(FIFO_sequence_item) scoreboard_fifo;
20
21     function new ( string name = "FIFO_scoreboard" , uvm_component parent = null );
22
23     super.new(name,parent);
24
25     endfunction
26
27     function void build_phase (uvm_phase phase);
28     super.build_phase(phase);
29     scoreboard_export = new("scoreboard_export",this);
30     scoreboard_fifo = new("scoreboard_fifo",this);
31     endfunction
32
33     function void connect_phase (uvm_phase phase);
34     super.connect_phase(phase);
35     scoreboard_export.connect(scoreboard_fifo.analysis_export);
36     endfunction
37
38     task run_phase (uvm_phase phase);
39     super.run_phase(phase);
40     forever begin
41         scoreboard_fifo.get(scoreboard_seq_item);
42         check_data(scoreboard_seq_item);
43     end
44     endtask
45
```

```

47 task reference_model (FIFO_sequence_item sb_seq_item) ;
48 if(sb_seq_item.rst_n) begin
49     if(sb_seq_item.wr_en &&!sb_seq_item.rd_en && count < 8) begin
50         mem.push_front(sb_seq_item.data_in);
51         count++;
52     end
53     else if(!sb_seq_item.wr_en && sb_seq_item.rd_en && count !=0 ) begin
54         data_out_ref=mem.pop_back;
55         count--;
56     end
57
58     else if(sb_seq_item.wr_en && sb_seq_item.rd_en && count > 0 && count < 8) begin
59         mem.push_front(sb_seq_item.data_in);
60         data_out_ref=mem.pop_back;
61         count=count;
62     end
63
64     else if (sb_seq_item.wr_en && sb_seq_item.rd_en && count == 8) begin
65         data_out_ref=mem.pop_back;
66         count--;
67     end
68     else if (sb_seq_item.wr_en && sb_seq_item.rd_en && count == 0) begin
69         mem.push_front(sb_seq_item.data_in);
70         count++;
71     end
72 end
73 else if(!sb_seq_item.rst_n) begin
74     mem.delete;
75     count = 0;
76 end
77
78 endtask

```

```

79
80 task check_data (FIFO_sequence_item sb_seq_item) ;
81 reference_model(sb_seq_item);
82 if(data_out_ref == sb_seq_item.data_out) begin
83     correct_count = correct_count + 1;
84 end
85 else begin
86     error_count = error_count + 1;
87     $display("Error: at %0t ns Expected data_out = 0x%0h ,data_out = 0x%0h ", $time, data_out_ref, sb_seq_item.data_out);
88     $display("Transaction details at %0t ns: rst_n = %0b, wr_en = %0b, rd_en = %0b, data_in = 0x%0h, data_out = 0x%0h, full = %0b, empty = %0b, wr_ack = %0b",
89             $time, sb_seq_item.rst_n, sb_seq_item.wr_en, sb_seq_item.rd_en, sb_seq_item.data_in, sb_seq_item.data_out, sb_seq_item.full, sb_seq_item.empty, sb_seq_item.wr_ack);
90 end
91
92
93 endtask
94
95 function void report_phase (uvm_phase phase);
96
97 super.report_phase(phase);
98 `uvm_info("report_phase",$sformatf("Total Number of Successful Transactions = %0d",correct_count),UVM_MEDIUM);
99 `uvm_info("report_phase",$sformatf("Total Number of Failed Transactions = %0d",error_count),UVM_MEDIUM);
100 endfunction
101
102
103 endclass
104
105 endpackage

```



## 7. Coverage Collector :

```

1  package FIFO_coverage_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4  import FIFO_sequence_item_pkg::*;
5
6  class FIFO_coverage extends uvm_component;
7  `uvm_component_utils(FIFO_coverage)
8
9  FIFO_sequence_item cov_seq_item;
10 uvm_analysis_export #(FIFO_sequence_item) cov_export;
11 uvm_tlm_analysis_fifo #(FIFO_sequence_item) cov_fifo;
12
13
14 covergroup FIFO_cvg ;
15
16 wr_cp:coverpoint cov_seq_item.wr_en {
17     |
18     |
19     |
20     |
21     |
22     |
23     |
24     |
25     bins wr_en_one = {1};
26     bins wr_en_zero = {0};
27 }
28
29 rd_cp:coverpoint cov_seq_item.rd_en {
30     |
31     |
32     |
33     |
34     |
35     |
36     |
37     |
38     bins rd_en_one = {1};
39     bins rd_en_zero = {0};
40 }
41
42 almostfull_cp: coverpoint cov_seq_item.almostfull {
43     |
44     |
45     |
46     |
47     |
48     |
49     |
50     |
51     bins almostfull_one = {1};
52     bins almostfull_zero = {0};
53 }
54

```



```

31  full_cp: coverpoint cov_seq_item.full {
32  | | | | | bins full_one = {1};
33  | | | | | bins full_zero = {0};
34  }
35
36  almostempty_cp: coverpoint cov_seq_item.almostempty {
37  | | | | | | | bins almostempty_one = {1};
38  | | | | | | | bins almostempty_zero = {0};
39  }
40
41  empty_cp: coverpoint cov_seq_item.empty {
42  | | | | | | | bins empty_one = {1};
43  | | | | | | | bins empty_zero = {0};
44  }
45
46  of_cp: coverpoint cov_seq_item.overflow {
47  | | | | | | | bins overflow_one = {1};
48  | | | | | | | bins overflow_zero = {0};
49  }
50
51  uf_cp: coverpoint cov_seq_item.underflow {
52  | | | | | | | bins underflow_one = {1};
53  | | | | | | | bins underflow_zero = {0};
54  }
55
56  wr_ack_cp: coverpoint cov_seq_item.wr_ack {
57  | | | | | | | bins wr_ack_one = {1};
58  | | | | | | | bins wr_ack_zero = {0};
59
60  }
61

```

```

62  wr_rd_almostfull_cp: cross wr_cp, rd_cp, almostfull_cp ;
63
64  wr_rd_full_cp: cross wr_cp, rd_cp, full_cp {
65  | | | | | | | illegal_bins no_write_read_full = binsof(full_cp.full_one) && binsof(rd_cp.rd_en_one) && binsof(wr_cp.wr_en_zero);
66  | | | | | | | illegal_bins write_read_full = binsof(full_cp.full_one) && binsof(rd_cp.rd_en_one) && binsof(wr_cp.wr_en_one);
67  | | | | | | | }
68
69  wr_rd_almostempty_cp: cross wr_cp, rd_cp, almostempty_cp;
70
71  wr_rd_empty_cp: cross wr_cp, rd_cp, empty_cp;
72
73  wr_rd_overflow_cp: cross wr_cp, rd_cp, of_cp {
74  | | | | | | | illegal_bins no_write_no_read_of = binsof(of_cp.overflow_one) && binsof(rd_cp.rd_en_zero) && binsof(wr_cp.wr_en_zero);
75  | | | | | | | illegal_bins no_write_read_of = binsof(of_cp.overflow_one) && binsof(rd_cp.rd_en_one) && binsof(wr_cp.wr_en_zero);
76  | | | | | | | }
77
78  wr_rd_underflow_cp: cross wr_cp, rd_cp, uf_cp {
79  | | | | | | | illegal_bins write_no_read_uf = binsof(uf_cp.underflow_one) && binsof(rd_cp.rd_en_zero) && binsof(wr_cp.wr_en_one);
80  | | | | | | | illegal_bins no_write_no_read_uf = binsof(uf_cp.underflow_one) && binsof(rd_cp.rd_en_zero) && binsof(wr_cp.wr_en_zero);
81  | | | | | | | }
82
83  wr_rd_ack_cp: cross wr_cp, rd_cp, wr_ack_cp {
84  | | | | | | | illegal_bins no_write_no_read_wr_ack = binsof(wr_ack_cp.wr_ack_one) && binsof(wr_cp.wr_en_zero) && binsof(rd_cp.rd_en_zero);
85  | | | | | | | illegal_bins no_write_read_wr_ack = binsof(wr_ack_cp.wr_ack_one) && binsof(wr_cp.wr_en_zero) && binsof(rd_cp.rd_en_one);
86  | | | | | | | }
87  endgroup

```

```
89  function new ( string name = "FIFO_coverage" , uvm_component parent = null );
90
91  super.new(name,parent);
92  FIFO_cvg=new();
93  endfunction
94
95  function void build_phase(uvm_phase phase) ;
96      super.build_phase(phase);
97      cov_export = new ("cov_export",this);
98      cov_fifo = new ("cov_fifo",this);
99  endfunction
100
101  function void connect_phase (uvm_phase phase);
102  super.connect_phase(phase);
103  cov_export.connect(cov_fifo.analysis_export);
104  endfunction
105
106  task run_phase (uvm_phase phase);
107
108  super.run_phase(phase);
109  forever begin
110      cov_fifo.get(cov_seq_item);
111      FIFO_cvg.start();
112      FIFO_cvg.sample();
113  end
114  endtask
115
116
117
118  endclass
119
120  endpackage
```

## 8. Sequence Item :

```
1  package FIFO_sequence_item_pkg;
2
3  import uvm_pkg::*;
4  `include "uvm_macros.svh"
5  import shared_pkg::*;
6  class FIFO_sequence_item extends uvm_sequence_item ;
7  `uvm_object_utils(FIFO_sequence_item)
8
9  rand bit rst_n;
10 rand bit [FIFO_WIDTH-1 :0] data_in;
11 rand bit wr_en,rd_en;
12 logic [FIFO_WIDTH-1 :0] data_out;
13 logic almostfull,full;
14 logic almostempty,empty;
15 logic overflow,underflow;
16 logic wr_ack;
17 int RD_EN_ON_DIST , WR_EN_ON_DIST;
18
19 // Constraints //
20
21 constraint Assert_reset_less_often {
22 | | | | | | | | rst_n dist {1:=98 ,0:=2};
23 }
24
25
26 constraint WRITE_ENABLE{
27 | | | | | | | | wr_en dist {1:=WR_EN_ON_DIST , 0:=100-WR_EN_ON_DIST};
28 }
29
30 constraint READ_ENABLE{
31 | | | | | | | | rd_en dist {1:=RD_EN_ON_DIST , 0:=100-RD_EN_ON_DIST};
32 }
33
34
35 constraint write_only {
36 | | | | | wr_en == 1;
37 | | | | | rd_en == 0;
38 | | | | | rst_n == 1;
39 }
40
41 constraint read_only {
42 | | | | | wr_en == 0;
43 | | | | | rd_en == 1;
44 | | | | | rst_n == 1;
45 }
46
47
48 function new ( string name = "FIFO_sequence_item" ,int RD_EN_ON_DIST = 30 , int WR_EN_ON_DIST = 70 );
49 super.new(name);
50 this.RD_EN_ON_DIST=RD_EN_ON_DIST;
51 this.WR_EN_ON_DIST=WR_EN_ON_DIST;
52
53 endfunction
54
55 function string convert2string ();
56 return $sformatf("%s rst_n = 0b%0b, wr_en = 0b%0b, rd_en = 0b%0b, data_in = 0b%0b, data_out = 0b%0b, almostfull = 0b%0b, full = 0b%0b,almostempty
57 endfunction
58
59 function string convert2string_stimulus ();
60 return $sformatf("rst_n = 0b%0b, wr_en = 0b%0b, rd_en = 0b%0b, data_in = 0b%0b",rst_n, wr_en, rd_en, data_in);
61 endfunction
```

```

55 function string convert2string ();
56 return $sprintf("%s rst_n = 0b%0b, wr_en = 0b%0b, rd_en = 0b%0b, data_in = 0b%0b, data_out = 0b%0b, almostfull = 0b%0b, full = 0b%0b, almostempty = 0b%0b, empty = 0b%0b, overfl
57 endfunction
58
59 function string convert2string_stimulus ();
60 return $sprintf("rst_n = 0b%0b, wr_en = 0b%0b, rd_en = 0b%0b, data_in = 0b%0b",rst_n, wr_en, rd_en, data_in);
61 endfunction
62
63
64
65 endclass
66
67
68 endpackage

```

## 9. Sequence:

```

1  package FIFO_sequence_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4  import FIFO_sequence_item_pkg::*;
5
6  class FIFO_reset_sequence extends uvm_sequence;
7  `uvm_object_utils(FIFO_reset_sequence)
8  FIFO_sequence_item seq_item;
9
10 function new ( string name = "FIFO_reset_sequence");
11
12 super.new(name);
13
14 endfunction
15
16
17 task body ;
18
19 seq_item = FIFO_sequence_item::type_id::create("seq_item");
20 start_item(seq_item);
21 seq_item.rst_n = 1'b0 ;
22 seq_item.data_in =16'b0 ;
23 seq_item.wr_en = 1'b0 ;
24 seq_item.rd_en = 1'b0 ;
25 finish_item(seq_item);
26
27 endtask
28
29 endclass
30

```



```

31  class FIFO_write_sequence extends uvm_sequence;
32  `uvm_object_utils(FIFO_write_sequence);
33  FIFO_sequence_item seq_item;
34
35  function new ( string name = "FIFO_write_sequence");
36
37  super.new(name);
38
39  endfunction
40
41  task body ;
42  repeat(1000) begin
43  seq_item = FIFO_sequence_item::type_id::create("seq_item");
44  start_item(seq_item);
45  seq_item.constraint_mode(0);
46  seq_item.write_only.constraint_mode(1);
47  assert(seq_item.randomize());
48  finish_item(seq_item);;
49
50  end
51
52  endtask
53
54  endclass
55

```

```

56
57  class FIFO_read_sequence extends uvm_sequence;
58  `uvm_object_utils(FIFO_read_sequence);
59  FIFO_sequence_item seq_item;
60
61  function new ( string name = "FIFO_read_sequence");
62
63  super.new(name);
64
65  endfunction
66
67  task body ;
68  repeat(1000) begin
69  seq_item = FIFO_sequence_item::type_id::create("seq_item");
70  start_item(seq_item);
71  seq_item.constraint_mode(0);
72  seq_item.read_only.constraint_mode(1);
73  assert(seq_item.randomize());
74  finish_item(seq_item);;
75  end
76  endtask
77
78  endclass

```



```
80
81  class FIFO_write_read_sequence extends uvm_sequence;
82  `uvm_object_utils(FIFO_write_read_sequence);
83  FIFO_sequence_item seq_item;
84
85  function new ( string name = "FIFO_write_read_sequence");
86
87  super.new(name);
88
89  endfunction
90
91
92  task body ;
93  repeat(10000) begin
94  seq_item = FIFO_sequence_item::type_id::create("seq_item");
95  start_item(seq_item);
96  seq_item.constraint_mode(1);
97  seq_item.write_only.constraint_mode(0);
98  seq_item.read_only.constraint_mode(0);
99  assert(seq_item.randomize());
100  finish_item(seq_item);;
101  end
102  endtask
103
104  endclass
105
106
107
108
109  endpackage
```

## 10.Sequencer:

```
1  package FIFO_sequencer_pkg;
2
3  import uvm_pkg::*;
4  `include "uvm_macros.svh"
5  import FIFO_sequence_item_pkg::*;
6
7  class FIFO_sequencer extends uvm_sequencer #(FIFO_sequence_item);
8  `uvm_component_utils(FIFO_sequencer);
9
10 function new ( string name = "FIFO_sequencer" , uvm_component parent = null );
11
12 super.new(name,parent);
13
14 endfunction
15
16 endclass
17
18 endpackage
```

## 11.Driver:

```
1  package FIFO_driver_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4  import FIFO_sequence_item_pkg::*;
5
6  class FIFO_driver extends uvm_driver #(FIFO_sequence_item) ;
7  `uvm_component_utils(FIFO_driver)
8  FIFO_sequence_item driver_seq_item;
9  virtual FIFO_Interface FIFO_vif;
10
11  function new ( string name = "FIFO_driver" , uvm_component parent = null );
12
13  super.new(name,parent);
14
15  endfunction
16
17  task run_phase (uvm_phase phase);
18  super.run_phase(phase);
19  forever begin
20      driver_seq_item = FIFO_sequence_item::type_id::create("driver_seq_item");
21      seq_item_port.get_next_item(driver_seq_item);
22      FIFO_vif.rst_n = driver_seq_item.rst_n ;
23      FIFO_vif.wr_en = driver_seq_item.wr_en ;
24      FIFO_vif.rd_en = driver_seq_item.rd_en ;
25      FIFO_vif.data_in = driver_seq_item.data_in ;
26      @(negedge FIFO_vif.clk);
27      seq_item_port.item_done();
28      `uvm_info("run_phase",driver_seq_item.convert2string(),UVM_HIGH);
29  end
30  endtask
31
32  endclass
33
34  endpackage
```

## 12. Monitor:

```
1  package FIFO_monitor_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4  import FIFO_sequence_item_pkg::*;
5
6  class FIFO_monitor extends uvm_monitor ;
7  `uvm_component_utils(FIFO_monitor)
8
9  FIFO_sequence_item mon_seq_item;
10 virtual FIFO_Interface FIFO_vif;
11 uvm_analysis_port #(FIFO_sequence_item) mon_ap;
12
13 function new ( string name = "FIFO_monitor" , uvm_component parent = null );
14
15 super.new(name,parent);
16
17 endfunction
18
19 function void build_phase(uvm_phase phase);
20     super.build_phase(phase);
21     mon_ap = new("mon_ap",this);
22 endfunction
23
24 task run_phase (uvm_phase phase);
25
26 super.run_phase(phase);
27 forever begin
28     mon_seq_item = FIFO_sequence_item::type_id::create("mon_seq_item");
29     @(negedge FIFO_vif.clk);
30     mon_seq_item.rst_n = FIFO_vif.rst_n ;
31     mon_seq_item.wr_en = FIFO_vif.wr_en ;
32     mon_seq_item.rd_en = FIFO_vif.rd_en ;
33     mon_seq_item.data_in = FIFO_vif.data_in ;
34     mon_seq_item.data_out = FIFO_vif.data_out ;
35     mon_seq_item.almostfull = FIFO_vif.almostfull ;
36     mon_seq_item.full = FIFO_vif.full ;
37     mon_seq_item.almostempty = FIFO_vif.almostempty ;
38     mon_seq_item.empty = FIFO_vif.empty ;
39     mon_seq_item.overflow = FIFO_vif.overflow ;
40     mon_seq_item.underflow = FIFO_vif.underflow ;
41     mon_seq_item.wr_ack = FIFO_vif.wr_ack ;
42     mon_ap.write(mon_seq_item);
43     `uvm_info("run_phase",$sformatf("Transaction Broadcasting %s",mon_seq_item.convert2string()),UVM_HIGH);
44 end
45
46 endtask
47
48
49
50 endclass
51
52 endpackage
```

## 13. Shared package:

```
1  package shared_pkg;
2  parameter FIFO_DEPTH = 8;
3  parameter FIFO_WIDTH = 16;
4  endpackage
```

## 7- Do File:

```
1 vlib work
2 vlog -f src_files.list +cover -covercells -l sim.log
3 vsim -voptargs=+acc work.FIFO_top -classdebug -uvmcontrol=all -cover
4 add wave -position insertpoint sim:/FIFO_top/FIFO_if/*
5 add wave -position insertpoint sim:/FIFO_top/FIFO_DUT/*
6 coverage save FIFO.ucdb -onexit
7 run -all
```

## src\_files:

```
1 FIFO.sv
2 shared_pkg.sv
3 FIFO_Interface.sv
4 FIFO_config_obj.sv
5 FIFO_sequence_item.sv
6 FIFO_sequencer.sv
7 FIFO_sequence.sv
8 FIFO_monitor.sv
9 FIFO_driver.sv
10 FIFO_coverage_collector.sv
11 FIFO_scoreboard.sv
12 FIFO_agent.sv
13 FIFO_env.sv
14 FIFO_test.sv
15 FIFO_top.sv
```

## 8- Coverage Report:

### 1. Code Coverage:

#### 1- Branch Coverage:

```
=====
=== Instance: /FIFO_top/FIFO_DUT
=== Design Unit: work.FIFO
=====
Branch Coverage:
  Enabled Coverage          Bins    Hits    Misses  Coverage
  -----
  Branches                  25     25       0    100.00%
```



## 2- Statement Coverage:

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

### 3- Condition Coverage:

Condition Coverage:				
Enabled Coverage	Bins	Covered	Misses	Coverage
-----	----	----	-----	-----
Conditions	24	24	0	100.00%

#### 4- Toggle Coverage:

```

5  === Design Unit: work.FIFO_Interface
6  =====
7  Toggle Coverage:
8      Enabled Coverage          Bins      Hits      Misses  Coverage
9  -----
10     Toggles                   86       86       0      100.00%
11
12  =====Toggle Details=====
13
14  Toggle Coverage for instance /FIFO_top/FIFO_if --
15
16      Node          1H->0L      0L->1H  "Coverage"
17      -----
18      almostempty    1          1      100.00
19      almostfull     1          1      100.00
20      clk            1          1      100.00
21      data_in[15-0]  1          1      100.00
22      data_out[15-0] 1          1      100.00
23      empty          1          1      100.00
24      full           1          1      100.00
25      overflow       1          1      100.00
26      rd_en          1          1      100.00
27      rst_n          1          1      100.00
28      underflow     1          1      100.00
29      wr_ack         1          1      100.00
30      wr_en          1          1      100.00
31
32  Total Node Count   =          43
33  Toggled Node Count =          43
34  Untoggled Node Count =          0
35
36  Toggle Coverage    =      100.00% (86 of 86 bins)

```

```
883 Toggle Coverage:
884   Enabled Coverage      Bins      Hits      Misses  Coverage
885   -----
886   Toggles              20       20        0    100.00%
887
888   =====Toggle Details=====
889
890 Toggle Coverage for instance /FIFO_top/FIFO_DUT --
891
892   | | | | | | | | | | Node      1H->0L      0L->1H      "Coverage"
893   | | | | | | | | | | -----
894   | | | | | | | | | | count[3-0]          1          1      100.00
895   | | | | | | | | | | rd_ptr[2-0]          1          1      100.00
896   | | | | | | | | | | wr_ptr[2-0]          1          1      100.00
897
898 Total Node Count      =          10
899 Toggled Node Count   =          10
900 Untoggled Node Count =           0
901
902 Toggle Coverage      =    100.00% (20 of 20 bins)
903
```

## 2. Functional Coverage:

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_ir
/FIFO_coverage_pkg/FIFO_coverage		100.00%						
TYPE FIFO_cvg		100.00%	100	100.00...		✓		auto(1)
CVP FIFO_cvg::wr_cp		100.00%	100	100.00...		✓		
CVP FIFO_cvg::rd_cp		100.00%	100	100.00...		✓		
CVP FIFO_cvg::almostfull_cp		100.00%	100	100.00...		✓		
CVP FIFO_cvg::full_cp		100.00%	100	100.00...		✓		
CVP FIFO_cvg::almostempty_cp		100.00%	100	100.00...		✓		
CVP FIFO_cvg::empty_cp		100.00%	100	100.00...		✓		
CVP FIFO_cvg::of_cp		100.00%	100	100.00...		✓		
CVP FIFO_cvg::uf_cp		100.00%	100	100.00...		✓		
CVP FIFO_cvg::wr_ack_cp		100.00%	100	100.00...		✓		
CROSS FIFO_cvg::wr_rd_almostfull_cp		100.00%	100	100.00...		✓		
CROSS FIFO_cvg::wr_rd_full_cp		100.00%	100	100.00...		✓		
CROSS FIFO_cvg::wr_rd_almostempty_cp		100.00%	100	100.00...		✓		
CROSS FIFO_cvg::wr_rd_empty_cp		100.00%	100	100.00...		✓		
CROSS FIFO_cvg::wr_rd_overflow_cp		100.00%	100	100.00...		✓		
CROSS FIFO_cvg::wr_rd_underflow_cp		100.00%	100	100.00...		✓		
CROSS FIFO_cvg::wr_rd_ack_cp		100.00%	100	100.00...		✓		

## Details:

1218	Covergroup Coverage:						
1219	Covergroups	1	na	na	100.00%		
1220	Coverpoints/Crosses	16	na	na	na		
1221	Covergroup Bins	66	66	0	100.00%		
1222							
1223	Covergroup		Metric	Goal	Bins	Status	
1224							
1225	TYPE /FIFO_coverage_pkg/FIFO_coverage/FIFO_cvg		100.00%	100	-	Covered	
1226	covered/total bins:		66	66	-		
1227	missing/total bins:		0	66	-		
1228	% Hit:		100.00%	100	-		
1229	Coverpoint wr_cp		100.00%	100	-	Covered	
1230	covered/total bins:		2	2	-		
1231	missing/total bins:		0	2	-		
1232	% Hit:		100.00%	100	-		
1233	bin wr_en_one		8019	1	-	Covered	
1234	bin wr_en_zero		3982	1	-	Covered	
1235	Coverpoint rd_cp		100.00%	100	-	Covered	
1236	covered/total bins:		2	2	-		
1237	missing/total bins:		0	2	-		
1238	% Hit:		100.00%	100	-		
1239	bin rd_en_one		4000	1	-	Covered	
1240	bin rd_en_zero		8001	1	-	Covered	
1241	Coverpoint almostfull_cp		100.00%	100	-	Covered	
1242	covered/total bins:		2	2	-		
1243	missing/total bins:		0	2	-		
1244	% Hit:		100.00%	100	-		
1245	bin almostfull_one		2476	1	-	Covered	
1246	bin almostfull_zero		9525	1	-	Covered	
1247	Coverpoint full_cp		100.00%	100	-	Covered	
1248	covered/total bins:		2	2	-		
1249							

1250	missing/total bins:	0	2	-	
1251	% Hit:	100.00%	100	-	
1252	bin full_one	4967	1	-	Covered
1253	bin full_zero	7034	1	-	Covered
1254	Coverpoint almostempty_cp	100.00%	100	-	Covered
1255	covered/total bins:	2	2	-	
1256	missing/total bins:	0	2	-	
1257	% Hit:	100.00%	100	-	
1258	bin almostempty_one	520	1	-	Covered
1259	bin almostempty_zero	11481	1	-	Covered
1260	Coverpoint empty_cp	100.00%	100	-	Covered
1261	covered/total bins:	2	2	-	
1262	missing/total bins:	0	2	-	
1263	% Hit:	100.00%	100	-	
1264	bin empty_one	1400	1	-	Covered
1265	bin empty_zero	10601	1	-	Covered
1266	Coverpoint of_cp	100.00%	100	-	Covered
1267	covered/total bins:	2	2	-	
1268	missing/total bins:	0	2	-	
1269	% Hit:	100.00%	100	-	
1270	bin overflow_one	4376	1	-	Covered
1271	bin overflow_zero	7625	1	-	Covered
1272	Coverpoint uf_cp	100.00%	100	-	Covered
1273	covered/total bins:	2	2	-	
1274	missing/total bins:	0	2	-	
1275	% Hit:	100.00%	100	-	
1276	bin underflow_one	1153	1	-	Covered
1277	bin underflow_zero	10848	1	-	Covered
1278	Coverpoint wr_ack_cp	100.00%	100	-	Covered
1279	covered/total bins:	2	2	-	
1280	missing/total bins:	0	2	-	
1281	% Hit:	100.00%	100	-	
1282	bin wr_ack_one	4157	1	-	Covered
1283	bin wr_ack_zero	7844	1	-	Covered
1284	Cross wr_rd_almostfull_cp	100.00%	100	-	Covered
1285	covered/total bins:	8	8	-	
1286	missing/total bins:	0	8	-	
1287	% Hit:	100.00%	100	-	
1288	Auto, Default and User Defined Bins:				
1289	bin <wr_en_zero,rd_en_zero,almostfull_zero>				
1290		1596	1	-	Covered
1291	bin <wr_en_one,rd_en_zero,almostfull_zero>				
1292		5555	1	-	Covered
1293	bin <wr_en_zero,rd_en_one,almostfull_zero>				
1294		1559	1	-	Covered
1295	bin <wr_en_one,rd_en_one,almostfull_zero>				
1296		815	1	-	Covered
1297	bin <wr_en_zero,rd_en_zero,almostfull_one>				
1298		478	1	-	Covered
1299	bin <wr_en_one,rd_en_zero,almostfull_one>				
1300		372	1	-	Covered
1301	bin <wr_en_zero,rd_en_one,almostfull_one>				
1302		349	1	-	Covered
1303	bin <wr_en_one,rd_en_one,almostfull_one>				
1304		1277	1	-	Covered
1305	Cross wr_rd_full_cp	100.00%	100	-	Covered
1306	covered/total bins:	6	6	-	
1307	missing/total bins:	0	6	-	

1308	% Hit:	100.00%	100	-	
1309	Auto, Default and User Defined Bins:				
1310	bin <wr_en_zero,rd_en_zero,full_zero>	1242	1	-	Covered
1311	bin <wr_en_zero,rd_en_one,full_zero>	1908	1	-	Covered
1312	bin <wr_en_one,rd_en_zero,full_zero>	1792	1	-	Covered
1313	bin <wr_en_one,rd_en_one,full_zero>	2092	1	-	Covered
1314	bin <wr_en_zero,rd_en_zero,full_one>	832	1	-	Covered
1315	bin <wr_en_one,rd_en_zero,full_one>	4135	1	-	Covered
1316	Illegal and Ignore Bins:				
1317	illegal_bin write_read_full	0	-	-	ZERO
1318	illegal_bin no_write_read_full	0	-	-	ZERO
1319	Cross wr_rd_almostempty_cp	100.00%	100	-	Covered
1320	covered/total bins:	8	8	-	
1321	missing/total bins:	0	8	-	
1322	% Hit:	100.00%	100	-	
1323	Auto, Default and User Defined Bins:				
1324	bin <wr_en_zero,rd_en_zero,almostempty_zero>				
1325		1978	1	-	Covered
1326	bin <wr_en_one,rd_en_zero,almostempty_zero>				
1327		5758	1	-	Covered
1328	bin <wr_en_zero,rd_en_one,almostempty_zero>				
1329		1861	1	-	Covered
1330	bin <wr_en_one,rd_en_one,almostempty_zero>				
1331		1884	1	-	Covered
1332	bin <wr_en_zero,rd_en_zero,almostempty_one>				
1333		96	1	-	Covered
1334	bin <wr_en_one,rd_en_zero,almostempty_one>				
1335		169	1	-	Covered
1336	bin <wr_en_zero,rd_en_one,almostempty_one>				
1336	bin <wr_en_zero,rd_en_one,almostempty_one>				
1337		47	1	-	Covered
1338	bin <wr_en_one,rd_en_one,almostempty_one>				
1339		208	1	-	Covered
1340	Cross wr_rd_empty_cp	100.00%	100	-	Covered
1341	covered/total bins:	8	8	-	
1342	missing/total bins:	0	8	-	
1343	% Hit:	100.00%	100	-	
1344	Auto, Default and User Defined Bins:				
1345	bin <wr_en_zero,rd_en_zero,empty_zero>	1929	1	-	Covered
1346	bin <wr_en_one,rd_en_zero,empty_zero>	5808	1	-	Covered
1347	bin <wr_en_zero,rd_en_one,empty_zero>	816	1	-	Covered
1348	bin <wr_en_one,rd_en_one,empty_zero>	2048	1	-	Covered
1349	bin <wr_en_zero,rd_en_zero,empty_one>	145	1	-	Covered
1350	bin <wr_en_one,rd_en_zero,empty_one>	119	1	-	Covered
1351	bin <wr_en_zero,rd_en_one,empty_one>	1092	1	-	Covered
1352	bin <wr_en_one,rd_en_one,empty_one>	44	1	-	Covered
1353	Cross wr_rd_overflow_cp	100.00%	100	-	Covered
1354	covered/total bins:	6	6	-	
1355	missing/total bins:	0	6	-	
1356	% Hit:	100.00%	100	-	
1357	Auto, Default and User Defined Bins:				
1358	bin <wr_en_zero,rd_en_zero,overflow_zero>				
1359		2074	1	-	Covered
1360	bin <wr_en_zero,rd_en_one,overflow_zero>				
1361		1908	1	-	Covered
1362	bin <wr_en_one,rd_en_zero,overflow_zero>				
1363		2523	1	-	Covered



1364		bin <wr_en_one,rd_en_one,overflow_zero>				
1365			1120	1	-	Covered
1366		bin <wr_en_one,rd_en_zero,overflow_one>				
1367			3404	1	-	Covered
1368		bin <wr_en_one,rd_en_one,overflow_one>				
1369			972	1	-	Covered
1370		Illegal and Ignore Bins:				
1371		illegal_bin no_write_read_uf	0		-	ZERO
1372		illegal_bin no_write_no_read_of	0		-	ZERO
1373		Cross wr_rd_underflow_cp	100.00%	100	-	Covered
1374		covered/total bins:	6	6	-	
1375		missing/total bins:	0	6	-	
1376		% Hit:	100.00%	100	-	
1377		Auto, Default and User Defined Bins:				
1378		bin <wr_en_zero,rd_en_zero,underflow_zero>				
1379			2074	1	-	Covered
1380		bin <wr_en_zero,rd_en_one,underflow_zero>				
1381			861	1	-	Covered
1382		bin <wr_en_one,rd_en_zero,underflow_zero>				
1383			5927	1	-	Covered
1384		bin <wr_en_one,rd_en_one,underflow_zero>				
1385			1986	1	-	Covered
1386		bin <wr_en_zero,rd_en_one,underflow_one>				
1387			1047	1	-	Covered
1388		bin <wr_en_one,rd_en_one,underflow_one>				
1389			106	1	-	Covered
1390		Illegal and Ignore Bins:				
1391		illegal_bin no_write_no_read_uf	0		-	ZERO
1392		illegal_bin write_no_read_uf	0		-	ZERO
1393		Cross wr_rd_ack_cp	100.00%	100	-	Covered
1394		covered/total bins:	6	6	-	
1395		missing/total bins:	0	6	-	
1396		% Hit:	100.00%	100	-	
1397		Auto, Default and User Defined Bins:				
1398		bin <wr_en_zero,rd_en_zero,wr_ack_zero>				
1399			2074	1	-	Covered
1400		bin <wr_en_zero,rd_en_one,wr_ack_zero>				
1401			1908	1	-	Covered
1402		bin <wr_en_one,rd_en_zero,wr_ack_zero>				
1403			3049	1	-	Covered
1404		bin <wr_en_one,rd_en_one,wr_ack_zero>				
1405			813	1	-	Covered
1406		bin <wr_en_one,rd_en_zero,wr_ack_one>				
1407			2878	1	-	Covered
1408		bin <wr_en_one,rd_en_one,wr_ack_one>				
1409			1279	1	-	Covered
1410		Illegal and Ignore Bins:				
1411		illegal_bin no_write_read_wr_ack	0		-	ZERO
1412		illegal_bin no_write_no_read_wr_ack	0		-	ZERO

### 3. Sequential Domain Coverage(Assertions):

```
38 =====
39 === Instance: /FIFO_top/FIFO_DUT/FIFO_SVA
40 === Design Unit: work.FIFO_assertions
41 =====
42
43 Assertion Coverage:
44 | Assertions                13      13      0  100.00%
45 |-----|
46 | Name                      File(Line)                      Failure    Pass
47 |                           |                               Count      Count
48 |-----|-----|
49 | /FIFO_top/FIFO_DUT/FIFO_SVA/wr_ack_a
50 | | | | | FIFO_assertions.sv(8)                                0          1
51 | /FIFO_top/FIFO_DUT/FIFO_SVA/overflow_a
52 | | | | | FIFO_assertions.sv(15)                               0          1
53 | /FIFO_top/FIFO_DUT/FIFO_SVA/underflow_a
54 | | | | | FIFO_assertions.sv(22)                               0          1
55 | /FIFO_top/FIFO_DUT/FIFO_SVA/wr_ptr_a
56 | | | | | FIFO_assertions.sv(31)                               0          1
57 | /FIFO_top/FIFO_DUT/FIFO_SVA/rd_ptr_a
58 | | | | | FIFO_assertions.sv(40)                               0          1
59 | /FIFO_top/FIFO_DUT/FIFO_SVA/count_inc_a
60 | | | | | FIFO_assertions.sv(49)                               0          1
61 | /FIFO_top/FIFO_DUT/FIFO_SVA/count_dec_a
62 | | | | | FIFO_assertions.sv(57)                               0          1
63 | /FIFO_top/FIFO_DUT/FIFO_SVA/count_no_change_a
64 | | | | | FIFO_assertions.sv(66)                               0          1
65 | /FIFO_top/FIFO_DUT/FIFO_SVA/comb_outputs/full_check
66 | | | | | FIFO_assertions.sv(76)                               0          1
67 | /FIFO_top/FIFO_DUT/FIFO_SVA/comb_outputs/almost_full_check
68 | | | | | FIFO_assertions.sv(80)                               0          1
69 | /FIFO_top/FIFO_DUT/FIFO_SVA/comb_outputs/empty_check
70 | | | | |
71 | /FIFO_top/FIFO_DUT/FIFO_SVA/count_dec_a
72 | | | | | FIFO_assertions.sv(57)                                0          1
73 | /FIFO_top/FIFO_DUT/FIFO_SVA/count_no_change_a
74 | | | | | FIFO_assertions.sv(66)                                0          1
75 | /FIFO_top/FIFO_DUT/FIFO_SVA/comb_outputs/full_check
76 | | | | | FIFO_assertions.sv(76)                                0          1
77 | /FIFO_top/FIFO_DUT/FIFO_SVA/comb_outputs/almost_full_check
78 | | | | | FIFO_assertions.sv(80)                                0          1
79 | /FIFO_top/FIFO_DUT/FIFO_SVA/comb_outputs/empty_check
80 | | | | | FIFO_assertions.sv(84)                                0          1
81 | /FIFO_top/FIFO_DUT/FIFO_SVA/comb_outputs/almostempty_check
82 | | | | | FIFO_assertions.sv(88)                                0          1
83 | /FIFO_top/FIFO_DUT/FIFO_SVA/reset_outputs/reset
84 | | | | | FIFO_assertions.sv(98)                                0          1
```

194	Directive Coverage:					
195	Directives	8	8	0	100.00%	
196						
197	DIRECTIVE COVERAGE:					
198	-----					
199	Name	Design	Design	Lang	File(Line)	Hits Status
200		Unit	UnitType			
201	-----					
202	/FIFO_top/FIFO_DUT/FIFO_SVA/wr_ack_c	FIFO_assertions	Verilog	SVA	FIFO_assertions.sv(9)	
203						2824 Covered
204	/FIFO_top/FIFO_DUT/FIFO_SVA/overflow_c	FIFO_assertions	Verilog	SVA	FIFO_assertions.sv(16)	
205						3634 Covered
206	/FIFO_top/FIFO_DUT/FIFO_SVA/underflow_c	FIFO_assertions	Verilog	SVA	FIFO_assertions.sv(23)	
207						1119 Covered
208	/FIFO_top/FIFO_DUT/FIFO_SVA/wr_ptr_c	FIFO_assertions	Verilog	SVA	FIFO_assertions.sv(32)	
209						4072 Covered
210	/FIFO_top/FIFO_DUT/FIFO_SVA/rd_ptr_c	FIFO_assertions	Verilog	SVA	FIFO_assertions.sv(41)	
211						2757 Covered
212	/FIFO_top/FIFO_DUT/FIFO_SVA/count_inc_c	FIFO_assertions	Verilog	SVA	FIFO_assertions.sv(50)	
213						2824 Covered
214	/FIFO_top/FIFO_DUT/FIFO_SVA/count_dec_c	FIFO_assertions	Verilog	SVA	FIFO_assertions.sv(58)	
215						839 Covered
216	/FIFO_top/FIFO_DUT/FIFO_SVA/count_no_change_c					
217		FIFO_assertions	Verilog	SVA	FIFO_assertions.sv(67)	
218						1166 Covered

## 9- Bugs Detected:

1- Output underflow is supposed to be sequential output while it was assigned as combinational output

Bug:

```
66 assign underflow = (empty && rd_en)? 1 : 0;
```

Fixed Code :

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

2-Output almostfull is asserted to be high when count = FIFO\_DEPTH -1 not FIFO\_DEPTH -2 (when reaching FIFO\_DEPTH -1 means we 've written FIFO\_DEPTH -1 times so there's only one write operation needed to be full)

Bug:

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

Fixed Design:

```
74 assign FIFO_if.almostfull = (count == FIFO_if.FIFO_DEPTH-1)? 1 : 0; // should be FIFO_if.FIFO_DEPTH-1 not FIFO_if.FIFO_DEPTH-2
```

3- The RTL of Design doesn't cover cases when write && read enables asserted while the fifo is empty or full.

Bug:

```
52  always @(posedge clk or negedge rst_n) begin
53      if (!rst_n) begin
54          count <= 0;
55      end
56      else begin
57          if ( ({wr_en, rd_en} == 2'b10) && !full)
58              count <= count + 1;
59          else if ( ({wr_en, rd_en} == 2'b01) && !empty)
60              count <= count - 1;
61      end
62  end
```

Fixed Design:

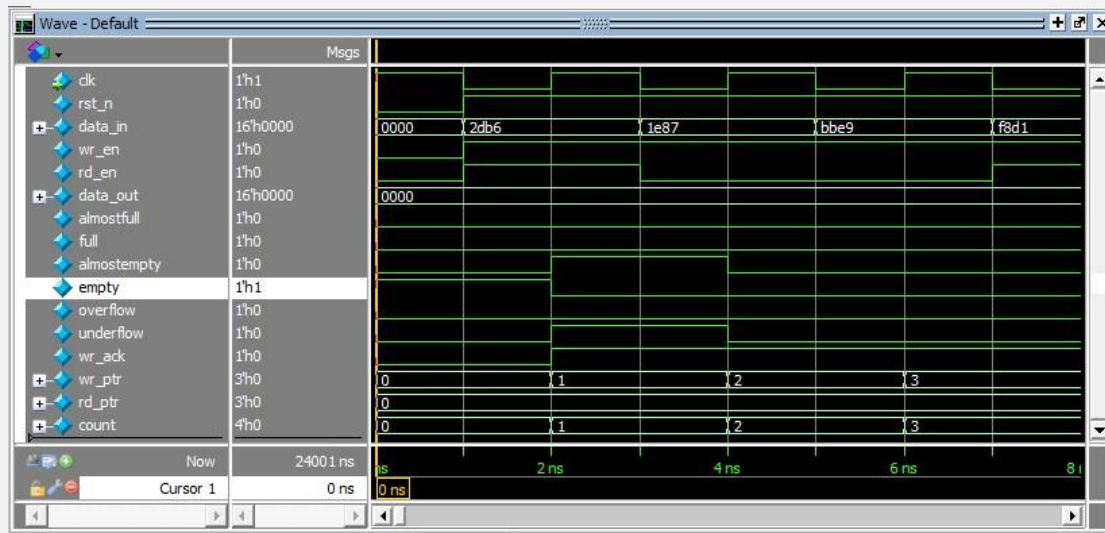
```
54  always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
55      if (!FIFO_if.rst_n) begin
56          count <= 0;
57      end
58      else begin
59          if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b10) && !FIFO_if.full)
60              count <= count + 1;
61          else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b01) && !FIFO_if.empty)
62              count <= count - 1;
63          else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.empty) begin
64              count <= count + 1;
65          end
66          else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.full)
67              count <= count - 1;
68      end
69  end
70
```

4- bug: when reset asserted overflow ,underflow, and wr\_Ack must be zero

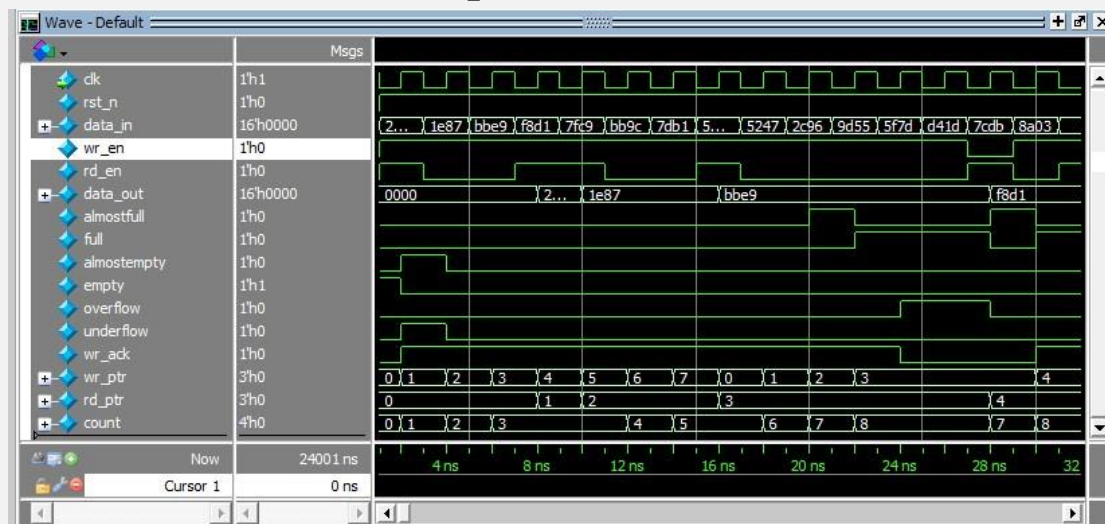


## 10- Waveform Snippets:

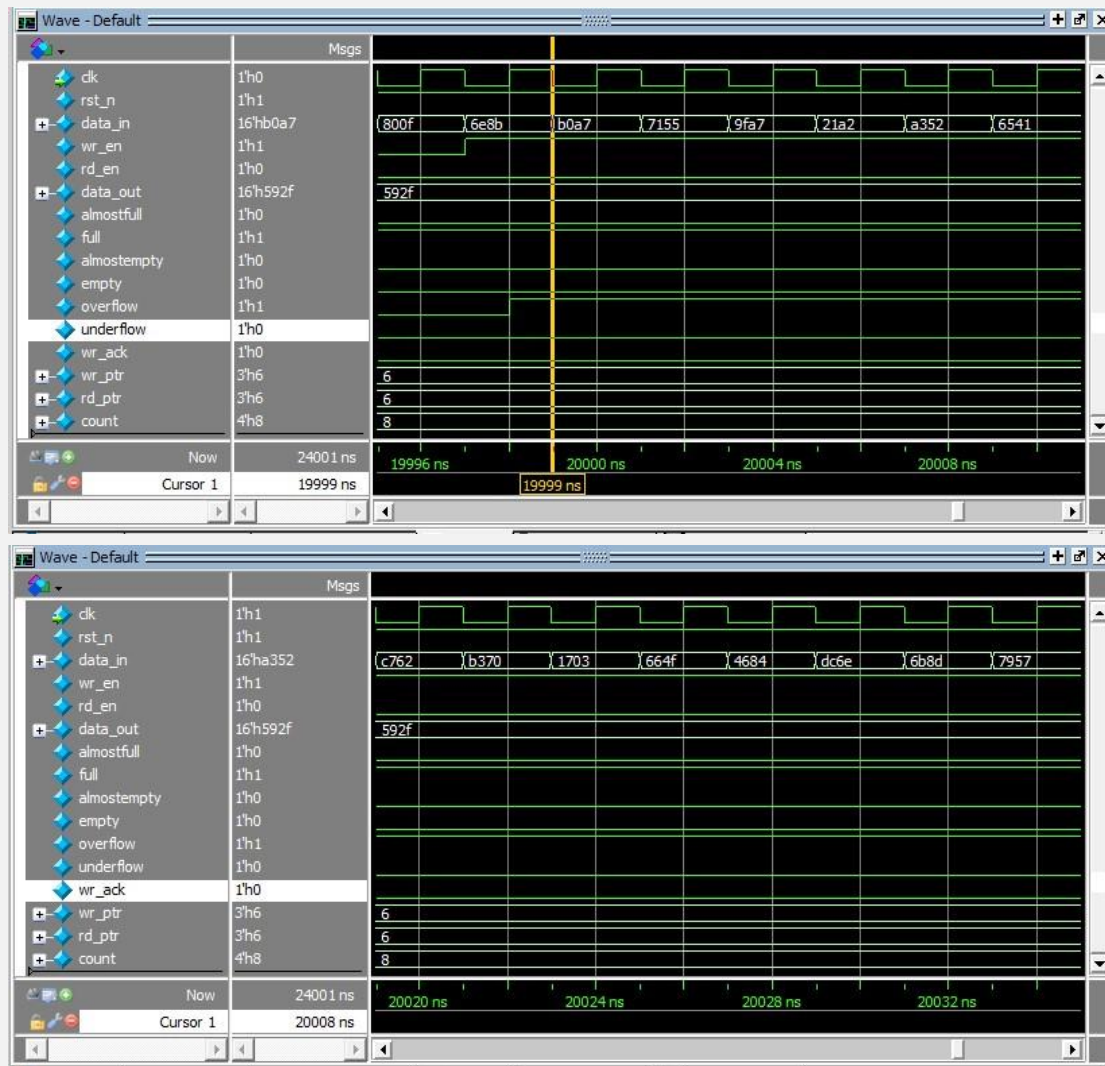
### 1.Reset Sequence: 0 – 1 ns



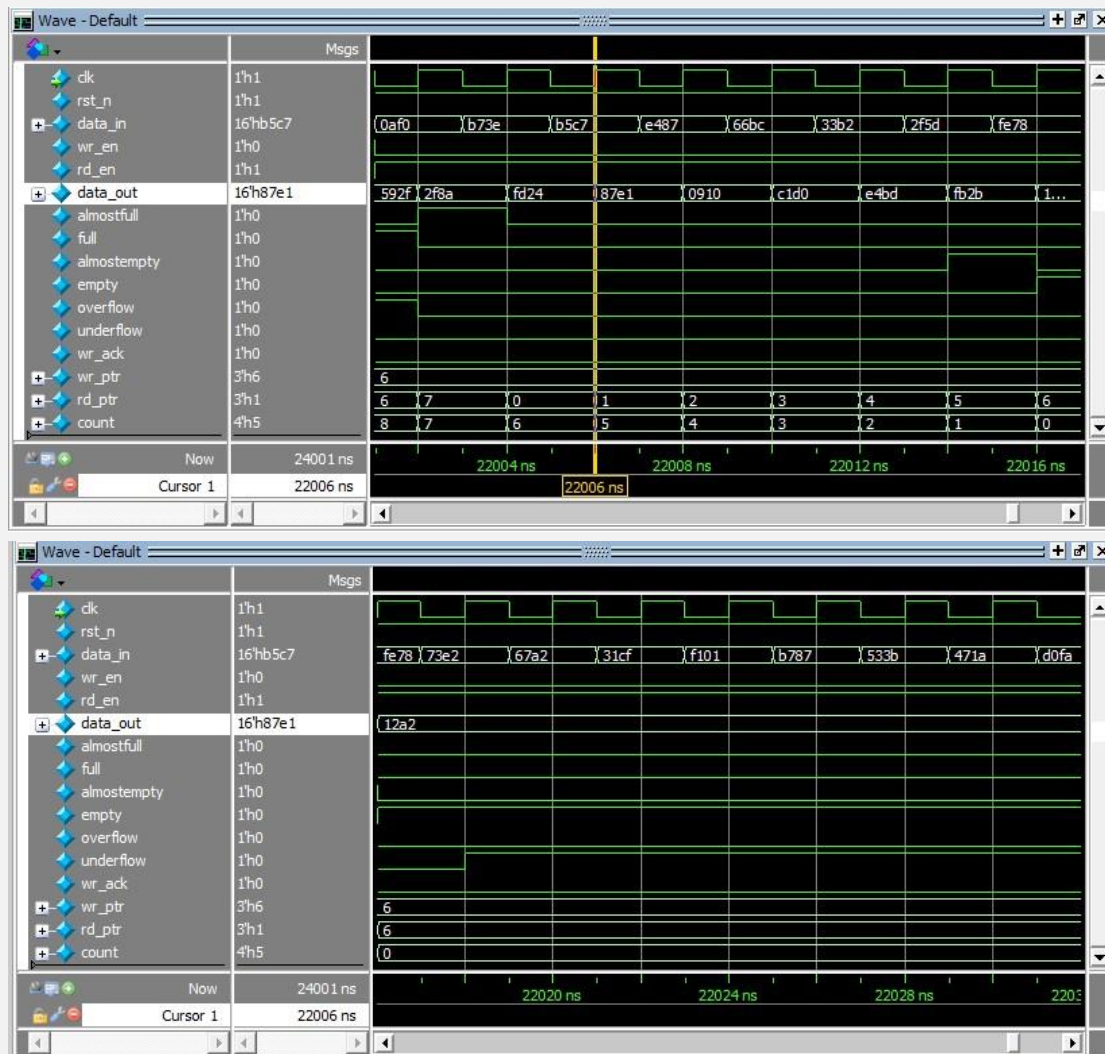
### 2. Write Read Sequence: 1- 20001 ns



### 3. Write Sequence : 20001 - 22001 ns



## 4. Read Sequence : 22001 – 24001 ns



## Transcript & UVM Testbench Summary :

```
Transcript
# *****
# UVM_INFO FIFO_test.sv(47) @ 1: uvm_test_top [run_phase] Reset Deasserted
# UVM_INFO FIFO_test.sv(50) @ 1: uvm_test_top [run_phase] Write Read Sequence has Started
# UVM_INFO FIFO_test.sv(52) @ 20001: uvm_test_top [run_phase] Write Read Sequence has Ended
# UVM_INFO FIFO_test.sv(55) @ 20001: uvm_test_top [run_phase] Write only Sequence has Started
# UVM_INFO FIFO_test.sv(57) @ 22001: uvm_test_top [run_phase] Write only Sequence has Ended
# UVM_INFO FIFO_test.sv(60) @ 22001: uvm_test_top [run_phase] Read only Sequence has Started
# UVM_INFO FIFO_test.sv(62) @ 24001: uvm_test_top [run_phase] Read only Sequence has Ended
# UVM_INFO verilog_svc/uvm-1.14/svc/base/uvm_object.svh(1267) @ 24001: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO FIFO_scoreboard.sv(99) @ 24001: uvm_test_top.env.sv [report_phase] Total Number of Successful Transactions = 12001
# UVM_INFO FIFO_scoreboard.sv(99) @ 24001: uvm_test_top.env.sv [report_phase] Total Number of Failed Transactions = 0

Transcript
# --- 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
```

```
Transcript
# ** Report counts by severity
# UVM_INFO : 14
# UVM_WARNING : 0
# UVM_ERROR : 0
# UVM_FATAL : 0
# ** Report counts by id
# [Questa UVM] 2
# [RNTST] 1
# [TEST_DONE] 1
# [report_phase] 2
# [run_phase] 8
# ** Note: $finish : C:/questasim64_2021.1/win64/./verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
```