

Synchronous FIFO – SV
Amira Atef Ismaeil El Komy

Table of Contents

1. Verification Plan.....	4
2. Code Snippets.....	5
2.1. Design.....	5
2.1.1. Bugs Found.....	6
2.2. Top Module.....	8
2.3. SVAs.....	9
2.4. Test Bench Module.....	10
2.5. Interface.....	11
2.6. Transaction Package.....	12
2.7. Monitor Module.....	13
2.8. Coverage Package.....	14
2.9. Scoreboard Package.....	15
2.10. Shared Package.....	17
3. Source Files List.....	18
4. Do File.....	18
5. Coverage Report Snippets.....	18
5.1. Branch Coverage Snippet.....	18
5.2. Toggle Coverage Snippet.....	18
5.3. Statement Coverage Snippet.....	19
5.4. Condition Coverage Snippet.....	19
5.5. Directive Coverage Snippet.....	19
5.6. Total Coverage Snippet.....	19
6. Functional Coverage Report Snippet.....	19
7. Assertions Report Snippet.....	20
8. Cover Directives Snippet.....	20

List of Figures

Figure 1: Sync. FIFO Verification Plan.....	4
Figure 2: Fixed Design Code	5
Figure 3: Fixed Design Code (Cont.).....	6
Figure 4: Bugs Found 1	6
Figure 5: Bugs Found 2	6
Figure 6: Bugs Found 3	7
Figure 7: Bugs Found 4	7
Figure 8: Bugs Found 5	7
Figure 9: Top Module Code	8
Figure 10: Properties Code	9
Figure 11: Assertions and Covers Code	9
Figure 12: Test Bench Module Code	10
Figure 13: Test Bench Module Code (Cont.)	11
Figure 14: Interface Code	11
Figure 15: Transaction Package Code.....	12
Figure 16: Monitor Module Code	13
Figure 17: Coverage Package Code.....	14
Figure 18: Coverage Package Code (Cont.).....	14
Figure 19: Scoreboard Package Code 1	15
Figure 20: Scoreboard Package Code 2	16
Figure 21: Scoreboard Package Code 3	17
Figure 22: Shared Package Code	17
Figure 23: List of Source Files	18
Figure 24: The Do File	18
Figure 25: Total Branch Coverage	18
Figure 26: Total Toggle Coverage	18
Figure 27: Total Statement Coverage	19
Figure 28: Total Condition Coverage	19
Figure 29: Total Directive Coverage	19
Figure 30: Total Coverage	19
Figure 31: Total Functional Coverage	19
Figure 32: Assertions	20
Figure 33: Cover Directives	20

1. Verification Plan

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

Figure 1: Sync. FIFO Verification Plan

2. Code Snippets

2.1. Design

```
10 module FIFO #(parameter FIFO_WIDTH = 16, FIFO_DEPTH = 8) (FIFO_if.DUT fifo_if);
11
12 logic clk, rst_n, wr_en, rd_en, wr_ack, overflow, full, empty, almostfull, almostempty, underflow;
13 logic [FIFO_WIDTH-1:0] data_in;
14 logic [FIFO_WIDTH-1:0] data_out;
15
16 // Interface signals
17 assign clk = fifo_if.clk;
18 assign rst_n = fifo_if.rst_n;
19 assign wr_en = fifo_if.wr_en;
20 assign rd_en = fifo_if.rd_en;
21 assign data_in = fifo_if.data_in;
22
23 assign fifo_if.data_out = data_out;
24 assign fifo_if.wr_ack = wr_ack;
25 assign fifo_if.overflow = overflow;
26 assign fifo_if.full = full;
27 assign fifo_if.empty = empty;
28 assign fifo_if.almostfull = almostfull;
29 assign fifo_if.almostempty = almostempty;
30 assign fifo_if.underflow = underflow;
31
32 localparam max_fifo_addr = $clog2(FIFO_DEPTH);
33
34 reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];
35
36 reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
37 reg [max_fifo_addr:0] count;
38
39 always @(posedge clk or negedge rst_n) begin
40     if (!rst_n) begin
41         wr_ptr <= 0;
42         wr_ack <= 0; //wr_ack and overflow were not reset
43         overflow <= 0;
44     end
45     else if (wr_en && count < FIFO_DEPTH) begin
46         mem[wr_ptr] <= data_in;
47         wr_ack <= 1;
48         wr_ptr <= wr_ptr + 1;
49         overflow <= 0; //overflow should be zero if wr_en and !full
50     end
51     else begin
52         wr_ack <= 0;
53         if (full && wr_en) begin // should be && not & for conditional coverage
54             overflow <= 1;
55         end
56         else begin
57             overflow <= 0;
58         end
59     end
60 end
61
62 always @(posedge clk or negedge rst_n) begin
63     if (!rst_n) begin
```

Figure 2: Fixed Design Code

```

63     if (!rst_n) begin
64         rd_ptr <= 0;
65         data_out <= 0; //data_out was not reset
66         underflow <= 0; // rst_n was not accounted for
67     end
68     else if (rd_en && count != 0) begin
69         data_out <= mem[rd_ptr];
70         rd_ptr <= rd_ptr + 1;
71         underflow <= 1'b0;
72     end
73     else if (rd_en) begin
74         underflow <= 1'b1;
75     end
76     else begin
77         underflow <= 1'b0;
78     end
79 end
80
81 always @(posedge clk or negedge rst_n) begin
82     if (!rst_n) begin
83         count <= 0;
84     end
85     else begin
86         if ({wr_en, rd_en} == 2'b11) begin //The possibility of rd & wr enable with either empty or full was not considered.
87             if (empty)
88                 count <= count + 1;
89             else if (full)
90                 count <= count - 1;
91         end
92         else if ({wr_en, rd_en} == 2'b10) && !full)
93             count <= count + 1;
94         else if (rd_en && !empty) // for conditional coverage
95             count <= count - 1;
96     end
97 end
98
99 assign full = (count == FIFO_DEPTH)? 1 : 0;
100 assign empty = (count == 0)? 1 : 0; // underflow was placed inside read always block
101 assign almostfull = (count == FIFO_DEPTH-1)? 1 : 0; // it was FIFO_DEPTH-2, almostfull means 1 element can be added
102 assign almostempty = (count == 1)? 1 : 0;
103
104 `ifdef SIM

```

Figure 3: Fixed Design Code (Cont.)

2.1.1. Bugs Found


<pre> 39 always @(posedge clk or negedge rst_n) begin 40 if (!rst_n) begin 41 wr_ptr <= 0; 42 wr_ack <= 0; //wr_ack and overflow were not reset 43 overflow <= 0; 44 end 45 else if (wr_en && count < FIFO_DEPTH) begin 46 mem[wr_ptr] <= data_in; 47 wr_ack <= 1; 48 wr_ptr <= wr_ptr + 1; 49 overflow <= 0; //overflow should be zero if wr_en and !full </pre>		<pre> 24 always @(posedge clk or negedge rst_n) begin 25 if (!rst_n) begin 26 wr_ptr <= 0; 27 end 28 else if (wr_en && count < FIFO_DEPTH) begin 29 mem[wr_ptr] <= data_in; 30 wr_ack <= 1; 31 wr_ptr <= wr_ptr + 1; 32 end 33 else begin 34 wr_ack <= 0; </pre>
---	---	--

Figure 4: Bugs Found 1

<pre> 51 else begin 52 wr_ack <= 0; 53 if (full && wr_en) begin // should be && not & for conditional coverage 54 overflow <= 1; 55 end 56 else begin 57 overflow <= 0; 58 end </pre>	<pre> 34 wr_ack <= 0; 35 if (full & wr_en) 36 overflow <= 1; 37 else 38 overflow <= 0; 39 end 40 end 41 </pre>
--	---

Figure 5: Bugs Found 2

<pre> 62 always @(posedge clk or negedge rst_n) begin 63 if (!rst_n) begin 64 rd_ptr <= 0; 65 data_out <= 0; //data_out was not reset 66 underflow <= 0; // rst_n was not accounted for 67 end 68 else if (rd_en && count != 0) begin 69 data_out <= mem[rd_ptr]; 70 rd_ptr <= rd_ptr + 1; 71 underflow <= 1'b0; 72 end 73 else if (rd_en) begin 74 underflow <= 1'b1; 75 end 76 else begin 77 underflow <= 1'b0; 78 end 79 end </pre>		<pre> 42 always @(posedge clk or negedge rst_n) begin 43 if (!rst_n) begin 44 rd_ptr <= 0; 45 end 46 else if (rd_en && count != 0) begin 47 data_out <= mem[rd_ptr]; 48 rd_ptr <= rd_ptr + 1; 49 end 50 end 51 52 53 54 55 56 57 58 59 </pre>
--	--	--

Figure 6: Bugs Found 3

<pre> 81 always @(posedge clk or negedge rst_n) begin 82 if (!rst_n) begin 83 count <= 0; 84 end 85 else begin 86 if ({wr_en, rd_en} == 2'b11) begin //The possibility of rd & wr enable with either 87 if (empty) 88 count <= count + 1; 89 else if (full) 90 count <= count - 1; 91 end 92 else if (({wr_en, rd_en} == 2'b10) && !full) 93 count <= count + 1; 94 else if (rd_en && !empty) // for conditional coverage 95 count <= count - 1; 96 end 97 end </pre>		<pre> 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 63 64 65 66 67 68 </pre>
---	--	--

Figure 7: Bugs Found 4

<pre> 99 assign full = (count == FIFO_DEPTH)? 1 : 0; 100 assign empty = (count == 0)? 1 : 0; // underflow was placed inside read always block 101 assign almostfull = (count == FIFO_DEPTH-1)? 1 : 0; // it was FIFO_DEPTH-2, almostfull mea 102 assign almostempty = (count == 1)? 1 : 0; 103 </pre>		<pre> 64 assign full = (count == FIFO_DEPTH)? 1 : 0; 65 assign empty = (count == 0)? 1 : 0; 66 assign underflow = (empty && rd_en)? 1 : 0; 67 assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0; 68 assign almostempty = (count == 1)? 1 : 0; </pre>
---	--	---

Figure 8: Bugs Found 5

2.2. Top Module

```
8  `timescale 1ns / 1ps
9
10 import shared_pkg::*;
11 module top();
12
13     ///////////////////////////////////////////////////
14     /////////////////////////////////////////////////// Clock Generator ///////////////////////////////////////////////////
15     ///////////////////////////////////////////////////
16
17     bit clk;
18
19     initial begin
20
21         // Start the clock
22         forever
23             #5 clk = ~clk; // Clock period of 10 time units
24     end
25
26     FIFO_if #(.FIFO_WIDTH(FIFO_WIDTH)) fifo_if (clk);
27
28     FIFO #(.FIFO_WIDTH(FIFO_WIDTH), .FIFO_DEPTH(FIFO_DEPTH)) DUT (fifo_if);
29
30     FIFO_tb tb (fifo_if);
31
32     FIFO_monitor mon (fifo_if);
33
34     // FIFO SVA (fifo_if);
35     // bind FIFO DUT SVA (fifo_if);
36
37     initial begin
38         $dumpfile("tb.vcd");
39         $dumpvars(0, top);
40     end
41 endmodule
```

Figure 9: Top Module Code

2.3. SVAs

```

10 module FIFO #(parameter FIFO_WIDTH = 16, FIFO_DEPTH = 8) (FIFO_if.DUT fifo_if);
104 `ifdef SIM
105     // Assertions and coverage for FIFO behavior
106     logic rst_deasserted;
107
108     always @(posedge clk or negedge rst_n) begin
109         if (!rst_n)
110             rst_deasserted <= 1'b0;
111         else
112             rst_deasserted <= 1'b1;
113     end
114
115     always @(posedge clk) begin
116         if (!rst_deasserted) begin
117             assert (data_out == 0 && empty && !almostfull && !almostempty && !underflow && !overflow && !full && !wr_ack && count == 0)
118                 else $error("Reset values incorrect at time %0t", $time);
119         end
120     end
121
122     property overflow_prop;
123     @(posedge clk) disable iff (~rst_n) (wr_en && full) | => overflow ;
124 endproperty
125
126     property underflow_prop;
127     @(posedge clk) disable iff (~rst_n) (rd_en && empty) | => underflow ;
128 endproperty
129
130     property wr_ack_prop;
131     @(posedge clk) disable iff (~rst_n) (wr_en && count < FIFO_DEPTH) | => wr_ack ;
132 endproperty
133
134     property full_prop;
135     @(posedge clk) disable iff (~rst_n) (count == FIFO_DEPTH) | -> full;
136 endproperty
137
138     property empty_prop;
139     @(posedge clk) disable iff (~rst_n) (count == 0) | -> empty;
140 endproperty
141
142     property almostfull_prop;
143     @(posedge clk) disable iff (~rst_n) (count == FIFO_DEPTH-1) | -> almostfull;
144 endproperty
145
146     property almostempty_prop;
147     @(posedge clk) disable iff (~rst_n) (count == 1) | -> almostempty;
148 endproperty
149
150     property count_wr_prop;
151     @(posedge clk) disable iff (~rst_n) ((wr_en && !full && !rd_en) || (wr_en && rd_en && empty)) | => (count == $past(count) + 1);
152 endproperty
153
154     property count_rd_prop;
155     @(posedge clk) disable iff (~rst_n) ((rd_en && !empty && !wr_en) || (wr_en && rd_en && full)) | => (count == $past(count) - 1);
156 endproperty

```

Figure 10: Properties Code

```

158     assert property (overflow_prop) else $error("Overflow occurred when FIFO is full");
159     assert property (underflow_prop) else $error("Underflow occurred when FIFO is empty");
160     assert property (wr_ack_prop) else $error("Write acknowledge not asserted when FIFO is not full");
161     assert property (full_prop) else $error("FIFO not full when count is equal to FIFO_DEPTH");
162     assert property (empty_prop) else $error("FIFO not empty when count is zero");
163     assert property (almostfull_prop) else $error("FIFO not almost full when count is equal to FIFO_DEPTH-1");
164     assert property (almostempty_prop) else $error("FIFO not almost empty when count is equal to 1");
165     assert property (count_wr_prop) else $error("Count not updated correctly during write operation");
166     assert property (count_rd_prop) else $error("Count not updated correctly during read operation");
167
168     cover property (overflow_prop);
169     cover property (underflow_prop);
170     cover property (wr_ack_prop);
171     cover property (full_prop);
172     cover property (empty_prop);
173     cover property (almostfull_prop);
174     cover property (almostempty_prop);
175     cover property (count_wr_prop);
176     cover property (count_rd_prop);
177 `endif

```

Figure 11: Assertions and Covers Code

2.4. Test Bench Module

```
8  import shared_pkg::*;
9  import FIFO_transaction_pkg::*;
10 module FIFO_tb(FIFO_if.TEST fifo_if);
11     // Declare the local variables
12     int i;
13     int data;
14
15     //////////////////////////////////////////
16     // Applying Stimulus on Inputs //
17     //////////////////////////////////////////
18
19     FIFO_transaction trns;
20
21     initial begin
22
23         // Initialize the FIFO interface
24         initialize();
25
26         trns = new();
27
28         // Generate test data
29         for (i = 0; i < 1000; i++) begin
30             assert(trns.randomize);
31             fifo_if.rst_n = trns.rst_n;
32             fifo_if.wr_en = trns.wr_en;
33             fifo_if.rd_en = trns.rd_en;
34             fifo_if.data_in = trns.data_in;
35             @(negedge fifo_if.clk);
36         end
37
38         // Finish the test after writing all data
39         test_finished = 1;
40     end
41
42     //////////////////////////////////////////
43     // TASKS //
44     //////////////////////////////////////////
45
46     // Signals Initialization //
47
48     task initialize;
49     begin
50         fifo_if.wr_en = 1'b1;
51         fifo_if.rd_en = 1'b0;
52     end
```

Figure 12: Test Bench Module Code

```

52         fifo_if.rd_en  = 1'b0;
53         fifo_if.data_in = 16'hFF_FF; // Example data
54         correct_count = 0;
55         error_count  = 0;
56         test_finished = 0;
57         reset();
58     end
59 endtask
60
61 /////////////////////////////////////////////////// RESET ///////////////////////////////////
62
63 task reset;
64     begin
65         fifo_if.rst_n = 1'b0;
66         repeat(2) @(negedge fifo_if.clk); // Hold reset for 2 cycles
67         fifo_if.rst_n = 1'b1;
68         fifo_if.wr_en = 1'b0; // Disable write enable after reset
69     end
70 endtask
71 endmodule

```

Figure 13: Test Bench Module Code (Cont.)

2.5. Interface

```

8 interface FIFO_if #(parameter FIFO_WIDTH = 16) (input bit clk);
9
10 ///////////////////////////////////////////////////
11 /////////////////////////////////////////////////// Interface Signal Declaration ///////////////////////////////////
12 ///////////////////////////////////////////////////
13
14 logic [FIFO_WIDTH-1:0] data_in;
15 logic rst_n, wr_en, rd_en;
16 logic [FIFO_WIDTH-1:0] data_out;
17 logic wr_ack, overflow;
18 logic full, empty, almostfull, almostempty, underflow;
19
20 ///////////////////////////////////////////////////
21 /////////////////////////////////////////////////// Modports ///////////////////////////////////
22 ///////////////////////////////////////////////////
23
24 modport TEST (
25     input clk, data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow,
26     output rst_n, wr_en, rd_en, data_in
27 );
28
29 modport DUT (
30     input clk, rst_n, wr_en, rd_en, data_in,
31     output data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow
32 );
33
34 modport MONITOR (
35     input clk, rst_n, wr_en, rd_en, data_in, data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow
36 );
37 endinterface

```

Figure 14: Interface Code

2.6. Transaction Package

```
8 package FIFO_transaction_pkg;
9 import shared_pkg::*;
10 class FIFO_transaction;
11     // FIFO transaction properties
12     rand logic rst_n;
13     rand logic wr_en;
14     rand logic rd_en;
15     randc logic [FIFO_WIDTH-1:0] data_in;
16     logic [FIFO_WIDTH-1:0] data_out;
17     logic wr_ack;
18     logic overflow;
19     logic full;
20     logic empty;
21     logic almostfull;
22     logic almostempty;
23     logic underflow;
24     int WR_EN_ON_DISTANCE = 70; // Distance between write enable signals
25     int RD_EN_ON_DISTANCE = 30; // Distance between read enable signals
26
27     constraint c_rst_n {
28         rst_n dist {1'b1 := 90, 1'b0 := 10}; // 90% chance of being 1, 10% chance of being 0
29     } // Reset is active low
30
31     constraint c_wr_en {
32         wr_en dist { 1'b0 := (100 - WR_EN_ON_DISTANCE), 1'b1 := WR_EN_ON_DISTANCE}; // Write enable signal
33     }
34
35     constraint c_rd_en {
36         rd_en dist { 1'b0 := (100 - RD_EN_ON_DISTANCE), 1'b1 := RD_EN_ON_DISTANCE}; // Read enable signal
37     }
38
39 endclass
40 endpackage
```

Figure 15: Transaction Package Code

2.7. Monitor Module

```
8 import shared_pkg::*;
9 import FIFO_transaction_pkg::*;
10 import FIFO_scoreboard_pkg::*;
11 import FIFO_coverage_pkg::*;
12
13 module FIFO_monitor(FIFO_if.MONITOR fifo_if);
14     FIFO_transaction trns;
15     FIFO_scoreboard sb;
16     FIFO_coverage cov;
17     initial begin
18         trns = new();
19         sb = new();
20         cov = new();
21
22         forever begin
23             @(negedge fifo_if.clk);
24             trns.rst_n = fifo_if.rst_n;
25             trns.wr_en = fifo_if.wr_en;
26             trns.rd_en = fifo_if.rd_en;
27             trns.data_in = fifo_if.data_in;
28
29             trns.data_out = fifo_if.data_out;
30             trns.wr_ack = fifo_if.wr_ack;
31             trns.overflow = fifo_if.overflow;
32             trns.full = fifo_if.full;
33             trns.empty = fifo_if.empty;
34             trns.almostfull = fifo_if.almostfull;
35             trns.almostempty = fifo_if.almostempty;
36             trns.underflow = fifo_if.underflow;
37             fork
38                 begin
39                     @(posedge fifo_if.clk);
40                     sb.check_data(trns);
41                 end
42                 begin
43                     cov.sample_data(trns);
44                 end
45             join
46             if (test_finished) begin
47                 $display("Test finished: Correct count = %0d, Error count = %0d", correct_count, error_count);
48                 $finish;
49             end
50         end
51     end
52 endmodule
```

Figure 16: Monitor Module Code

2.8. Coverage Package

```

8  package FIFO_coverage_pkg;
9
10  import FIFO_transaction_pkg::*;
11
12  class FIFO_coverage;
13      FIFO_transaction F_cvg_txn;
14
15      covergroup FIFO_cvr_grp;
16          // Coverpoints for FIFO transaction signals
17          rst_n_c:    coverpoint F_cvg_txn.rst_n;
18          wr_en_c:    coverpoint F_cvg_txn.wr_en;
19          rd_en_c:    coverpoint F_cvg_txn.rd_en;
20          // data_in_c:    coverpoint F_cvg_txn.data_in {option.cross_auto_bin_max = 0;};
21          // data_out_c:   coverpoint F_cvg_txn.data_out {option.cross_auto_bin_max = 0;};
22          wr_ack_c:   coverpoint F_cvg_txn.wr_ack;
23          overflow_c: coverpoint F_cvg_txn.overflow;
24          full_c:     coverpoint F_cvg_txn.full;
25          empty_c:    coverpoint F_cvg_txn.empty;
26          almostfull_c: coverpoint F_cvg_txn.almostfull;
27          almostempty_c: coverpoint F_cvg_txn.almostempty;
28          underflow_c: coverpoint F_cvg_txn.underflow;
29
30          // Cross coverage for write and read enable signals
31
32          wr_ack_cross: cross wr_en_c, rd_en_c, wr_ack_c {
33              ignore_bins read_wr_ack = binsof(wr_en_c) intersect {0} && binsof(rd_en_c) intersect {1} && binsof(wr_ack_c) intersect {1};
34              ignore_bins write_read_wr_ack = binsof(wr_en_c) intersect {0} && binsof(wr_ack_c) intersect {1};
35          }
36
37          overflow_cross: cross wr_en_c, rd_en_c, overflow_c {
38              ignore_bins write_overflow = binsof(wr_en_c) intersect {0} && binsof(overflow_c) intersect {1};
39          }
40
41          full_cross: cross wr_en_c, rd_en_c, full_c {
42              ignore_bins write_full = binsof(wr_en_c) intersect {0} && binsof(full_c) intersect {1};
43              ignore_bins read_full = binsof(rd_en_c) intersect {1} && binsof(full_c) intersect {1};
44          }
45
46          empty_cross: cross wr_en_c, rd_en_c, empty_c {
47              ignore_bins read_empty = binsof(rd_en_c) intersect {0} && binsof(empty_c) intersect {1};
48          }
49
50          almostfull_cross: cross wr_en_c, rd_en_c, almostfull_c {
51              ignore_bins read_write_almostfull = binsof(wr_en_c) intersect {0} && binsof(rd_en_c) intersect {1} && binsof(almostfull_c) intersect {1};
52              ignore_bins write_almostfull = binsof(wr_en_c) intersect {0} && binsof(almostfull_c) intersect {1};
53          }
54
55          almostempty_cross: cross wr_en_c, rd_en_c, almostempty_c {
56              ignore_bins read_almostempty = binsof(rd_en_c) intersect {0} && binsof(almostempty_c) intersect {1};
57              ignore_bins write_read_almostempty = binsof(rd_en_c) intersect {0} && binsof(wr_en_c) intersect {1} && binsof(almostempty_c) intersect {1};
58          }
59
60          underflow_cross: cross wr_en_c, rd_en_c, underflow_c {
61              ignore_bins read_underflow = binsof(rd_en_c) intersect {0} && binsof(underflow_c) intersect {1};

```

Figure 17: Coverage Package Code

```

62          ignore_bins write_underflow = binsof(wr_en_c) intersect {1} && binsof(underflow_c) intersect {1};
63      }
64  endgroup : FIFO_cvr_grp
65
66  // Constructor
67  function new();
68      F_cvg_txn = new();
69      FIFO_cvr_grp = new();
70  endfunction
71
72  // Sample data for coverage
73  function void sample_data(FIFO_transaction F_txn);
74      // Implement coverage sampling logic here
75      F_cvg_txn = F_txn;
76      FIFO_cvr_grp.sample();
77  endfunction
78
79  endclass
80
81  endpackage

```

Figure 18: Coverage Package Code (Cont.)

2.9. Scoreboard Package

```
8 package FIFO_scoreboard_pkg;
9
10 import shared_pkg::*;
11 import FIFO_transaction_pkg::*;
12
13 class FIFO_scoreboard;
14     // FIFO transaction object
15     FIFO_transaction F_sb_txn;
16     bit [FIFO_WIDTH-1:0] data_out_ref;
17     bit wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;
18
19     localparam max_fifo_addr = $clog2(FIFO_DEPTH);
20     logic [FIFO_WIDTH-1:0] mem_ref [FIFO_DEPTH-1:0];
21     logic [max_fifo_addr-1:0] wr_ptr, rd_ptr;
22     logic [max_fifo_addr:0] count;
23
24     // Constructor
25     function new();
26         F_sb_txn = new();
27     endfunction
28
29     // Method to check FIFO status and update scoreboard
30     function void check_data(FIFO_transaction F_txn);
31         F_sb_txn = F_txn;
32         fifo_reference_model(F_sb_txn.rst_n, F_sb_txn.wr_en, F_sb_txn.rd_en, F_sb_txn.data_in);
33         if (F_sb_txn.data_out != data_out_ref) begin
34             error_count++;
35             $display("[%0t] Data mismatch: Expected %0h, Got %0h", $time, data_out_ref, F_sb_txn.data_out);
36         end else begin
37             correct_count++;
38         end
39         `ifndef SIM
40             if (F_sb_txn.wr_ack != wr_ack_ref) begin
41                 error_count++;
42                 $display("Write acknowledgment mismatch: Expected %0b, Got %0b", wr_ack_ref, F_sb_txn.wr_ack);
43             end
44             else begin
45                 correct_count++;
46             end
47
48             if (F_sb_txn.overflow != overflow_ref) begin
49                 error_count++;
50                 $display("Overflow mismatch: Expected %0b, Got %0b", overflow_ref, F_sb_txn.overflow);
51             end
52             else begin
53                 correct_count++;
54             end
55
56             if (F_sb_txn.full != full_ref) begin
57                 error_count++;
58                 $display("Full status mismatch: Expected %0b, Got %0b", full_ref, F_sb_txn.full);
59             end
60             else begin
61                 correct_count++;
62             end
63         end
64     endfunction
65 endclass
```

Figure 19: Scoreboard Package Code 1

```

62         end
63
64         if (F_sb_txn.empty != empty_ref) begin
65             error_count++;
66             $display("Empty status mismatch: Expected %0b, Got %0b", empty_ref, F_sb_txn.empty);
67         end
68         else begin
69             correct_count++;
70         end
71
72         if (F_sb_txn.almostfull != almostfull_ref) begin
73             error_count++;
74             $display("Almost full status mismatch: Expected %0b, Got %0b", almostfull_ref, F_sb_txn.almostfull);
75         end
76         else begin
77             correct_count++;
78         end
79
80         if (F_sb_txn.almostempty != almostempty_ref) begin
81             error_count++;
82             $display("Almost empty status mismatch: Expected %0b, Got %0b", almostempty_ref, F_sb_txn.almostempty);
83         end
84         else begin
85             correct_count++;
86         end
87
88         if (F_sb_txn.underflow != underflow_ref) begin
89             error_count++;
90             $display("Underflow status mismatch: Expected %0b, Got %0b", underflow_ref, F_sb_txn.underflow);
91         end
92         else begin
93             correct_count++;
94         end
95     `endif
96 endfunction
97
98 function void fifo_reference_model(
99     input bit rst_n,
100     input bit wr_en,
101     input bit rd_en,
102     input bit [FIFO_WIDTH-1:0] data_in
103 );
104     // Reference model logic to generate expected values
105     if (!rst_n) begin
106         data_out_ref = '0;
107         count = '0;
108         wr_ptr = '0;
109         rd_ptr = '0;
110         wr_ack_ref = 1'b0;
111         overflow_ref = 1'b0;
112         full_ref = 1'b0;
113         empty_ref = 1'h1;

```

Figure 20: Scoreboard Package Code 2


```

113     empty_ref = 1'b1;
114     almostfull_ref = 1'b0;
115     almostempty_ref = 1'b0;
116     underflow_ref = rd_en ? 1'b1 : 1'b0;
117 end
118 else begin
119     fork
120     begin
121         if (wr_en && count < FIFO_DEPTH) begin
122             mem_ref[wr_ptr] = data_in;
123             wr_ptr++;
124             wr_ack_ref = 1'b1;
125         end
126         else begin
127             wr_ack_ref = 1'b0;
128         end
129     end
130     begin
131         if (rd_en && count > 0) begin
132             data_out_ref = mem_ref[rd_ptr];
133             rd_ptr++;
134             empty_ref = (count == 0) ? 1'b1 : 1'b0;
135         end
136     end
137     join
138     if ({wr_en, rd_en} == 2'b11) begin
139         if (empty_ref)
140             count++;
141         else if (full_ref)
142             count--;
143     end
144     else if ({wr_en, rd_en} == 2'b10) && !full_ref
145         count++;
146     else if ({wr_en, rd_en} == 2'b01) && !empty_ref
147         count--;
148     full_ref = (count == FIFO_DEPTH) ? 1 : 0;
149     empty_ref = (count == 0) ? 1 : 0;
150     almostfull_ref = (count == FIFO_DEPTH-1) ? 1 : 0;
151     almostempty_ref = (count == 1) ? 1 : 0;
152     underflow_ref = (empty_ref) ? 1 : 0;
153 end
154 endfunction
155 endclass : FIFO_scoreboard
156
157 endpackage

```

Figure 21: Scoreboard Package Code 3

2.10. Shared Package

```

8  package shared_pkg;
9      bit test_finished; // Flag to indicate if the test is finished
10
11      //////////////////////////////////////
12      ////////////////////////////////////// Counters //////////////////////////////////////
13      //////////////////////////////////////
14
15      int correct_count, error_count;
16
17      //////////////////////////////////////
18      ////////////////////////////////////// Parameters //////////////////////////////////////
19      //////////////////////////////////////
20
21      parameter int FIFO_WIDTH = 16; // Width of the FIFO data bus
22      parameter int FIFO_DEPTH = 8; // Depth of the FIFO
23  endpackage

```

Figure 22: Shared Package Code

3. Source Files List

```
1 FIFO.sv
2 FIFO_if.sv
3 shared_pkg.sv
4 FIFO_transaction_pkg.sv
5 FIFO_scoreboard_pkg.sv
6 FIFO_coverage_pkg.sv
7 FIFO_monitor.sv
8 FIFO_tb.sv
9 top.sv
```

Figure 23: List of Source Files

4. Do File

```
1 vlib work
2 vlog -f src_files.list +define+SIM +cover -covercells
3 vsim -voptargs=+acc work.top -cover
4 add wave /top/fifo_if/*
5 coverage save FIFO_SV.ucdb -onexit -du work.FIFO
6 vcover report FIFO_SV.ucdb -details -annotate -all -output FIFO_SV_cvr_rpt.txt
7 run -all
```

Figure 24: The Do File

5. Coverage Report Snippets

5.1. Branch Coverage Snippet

```
34 Branch Coverage:
35   Enabled Coverage          Bins    Hits    Misses  Coverage
36   -----
37   Branches                  30     30      0    100.00%
38
```

Figure 25: Total Branch Coverage

5.2. Toggle Coverage Snippet

```
557 Toggle Coverage:
558   Enabled Coverage          Bins    Hits    Misses  Coverage
559   -----
560   Toggles                   108    108      0    100.00%
```

Figure 26: Total Toggle Coverage

5.3. Statement Coverage Snippet

332	Statement Coverage:				
333	Enabled Coverage	Bins	Hits	Misses	Coverage
334	-----	----	----	-----	-----
335	Statements	39	39	0	100.00%

Figure 27: Total Statement Coverage

5.4. Condition Coverage Snippet

152	Condition Coverage:				
153	Enabled Coverage	Bins	Covered	Misses	Coverage
154	-----	----	----	-----	-----
155	Conditions	17	17	0	100.00%

Figure 28: Total Condition Coverage

5.5. Directive Coverage Snippet

315	Directive Coverage:				
316	Directives	9	9	0	100.00%

Figure 29: Total Directive Coverage

5.6. Total Coverage Snippet

636	Total Coverage By Instance (filtered view): 100.00%
-----	---

Figure 30: Total Coverage

6. Functional Coverage Report Snippet

31	Covergroup Coverage:				
32	Covergroups	1	na	na	100.00%
33	Coverpoints/Crosses	17	na	na	na
34	Covergroup Bins	60	60	0	100.00%

Figure 31: Total Functional Coverage

7. Assertions Report Snippet

8	Assertion Coverage:				
9	Assertions	10	10	0	100.00%
10	-----				
11	Name	File(Line)	Failure	Pass	
12			Count	Count	
13	-----				
14	/top/DUT/assert__count_rd_prop				
15		FIFO.sv(166)	0	1	
16	/top/DUT/assert__count_wr_prop				
17		FIFO.sv(165)	0	1	
18	/top/DUT/assert__almostempty_prop				
19		FIFO.sv(164)	0	1	
20	/top/DUT/assert__almostfull_prop				
21		FIFO.sv(163)	0	1	
22	/top/DUT/assert__empty_prop				
23		FIFO.sv(162)	0	1	
24	/top/DUT/assert__full_prop				
25		FIFO.sv(161)	0	1	
26	/top/DUT/assert__wr_ack_prop				
27		FIFO.sv(160)	0	1	
28	/top/DUT/assert__underflow_prop				
29		FIFO.sv(159)	0	1	
30	/top/DUT/assert__overflow_prop				
31		FIFO.sv(158)	0	1	
32	/top/DUT/#ublk#306607#116/immed__117				
33		FIFO.sv(117)	0	1	

Figure 32: Assertions

8. Cover Directives Snippet

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
▶ /top/DUT/cover__count_rd_prop	SVA	✓	Off	100	1	Unli...	1	100%	██████████	✓	0	0	0 ps	0
▶ /top/DUT/cover__count_wr_prop	SVA	✓	Off	370	1	Unli...	1	100%	██████████	✓	0	0	0 ps	0
▶ /top/DUT/cover__almostempty_pro...	SVA	✓	Off	184	1	Unli...	1	100%	██████████	✓	0	0	0 ps	0
▶ /top/DUT/cover__almostfull_prop	SVA	✓	Off	106	1	Unli...	1	100%	██████████	✓	0	0	0 ps	0
▶ /top/DUT/cover__empty_prop	SVA	✓	Off	158	1	Unli...	1	100%	██████████	✓	0	0	0 ps	0
▶ /top/DUT/cover__full_prop	SVA	✓	Off	95	1	Unli...	1	100%	██████████	✓	0	0	0 ps	0
▶ /top/DUT/cover__wr_ack_prop	SVA	✓	Off	496	1	Unli...	1	100%	██████████	✓	0	0	0 ps	0
▶ /top/DUT/cover__underflow_prop	SVA	✓	Off	50	1	Unli...	1	100%	██████████	✓	0	0	0 ps	0
▶ /top/DUT/cover__overflow_prop	SVA	✓	Off	62	1	Unli...	1	100%	██████████	✓	0	0	0 ps	0

Figure 33: Cover Directives