

# SV Project - Synchronous FIFO

By: Amr Ayman Mohamed Abdo

## Contents

Verification Plan .....	2
Verification Requirements Document Snippet .....	2
Code Snippets .....	3
FIFO_transaction_pkg .....	3
FIFO_coverage_pkg .....	4
FIFO_monitor.....	4
FIFO_scoreboard_pkg .....	5
FIFO_ref (reference model) .....	6
FIFO_tb.....	7
FIFO_if.....	8
FIFO_top .....	9
run_FIFO (Do File) .....	10
src_files.list.....	10
Detected Bugs .....	11
DUT after fixing bugs and adding assertions .....	14
QuestaSim Snippets .....	15
Wave colour guidelines.....	15
Snippets.....	15
Coverage Reports .....	18
Code Coverage Report .....	18
Functional Coverage Report.....	19
Sequential Domain Coverage Report.....	20

# Verification Plan

1. Test writing operation by performing writes more often than reads. (**Only\_Write**)
2. Test reading operation by performing reads more often than writes. (**Only\_Read**)
3. Test the FIFO with multiple concurrent writers and readers to ensure data integrity and proper flag behavior. (**Concurrent\_Write\_Read**)
4. Test simultaneous write-read operation by performing reads and writes with equal distribution probability. (**Simultaneous\_Read\_Write**)
5. Fill the FIFO to its maximum capacity and attempt more writes to ensure the `full` and `almostfull` flags are set and `overflow` is triggered. (**Write\_Full**)
6. Completely empty the FIFO and attempt more reads to check if the `empty` and `almostempty` flags are set and `underflow` is triggered. (**Read\_Empty**)
7. With a full FIFO, perform a simultaneous write and read to check if the FIFO correctly updates the `full` flag and does not set the `overflow`. (**Simultaneous\_Read\_Write\_Full**)
8. With an empty FIFO, perform a simultaneous read and write and verify the correct behavior of the `empty` flag and does not set the `underflow`. (**Simultaneous\_Read\_Write\_Empty**)
9. On reset, verify that the FIFO is cleared and all flags are in the correct initial state. (**Reset**)
10. Perform random sequences of reads and writes to test the FIFO's robustness. (**Read\_Write\_Randomized**)
11. Test flags stability when neither reading nor writing is taking place. (**Flags\_Stability**)

## Verification Requirements Document Snippet

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
Only_Write	Write unique values to the FIFO while reading is disabled	constraint rd_en to be zero while randomizing wr_en to be high most of the time. Constraint data_in to have unique values.	Included in <b>write_seq</b> that ensures that 8 consecutive writes have taken place	Outputs are checked against reference model + <b>assert_full, assert_almostfull &amp; assert_wr_ack</b>
Only_Read	Read the previously written values to the FIFO while writing is disabled	constraint wr_en to be zero while randomizing rd_en to be high most of the time.	Included in <b>read_seq</b> that ensures that 8 consecutive reads have taken place	Outputs are checked against reference model + <b>assert_empty &amp; assert_almostempty</b>
Concurrent_Write_Read	Loop containing a sequence of writes followed by another sequence of reads	constraint rd_en to be zero while randomizing wr_en to be high most of the time while writing and vice versa	Included in <b>empty_to_not</b> and <b>full_to_not</b> that ensure that a writing sequence took place after a reading sequence and vice versa	Outputs are checked against reference model
Simultaneous_Read_Write	Read and Write simultaneously and check for the flags remaining low.	constraint rd_en and wr_en to be high simultaneously most of the time.	Included in <b>write_read_simit</b> cross bin that covers read cross write	Outputs are checked against reference model
Write_Full	Write to a full FIFO and check for the overflow to be deasserted when rd_en is asserted	When the FIFO is full constraint wr_en to be high most of the time and rd_en to be low most of the time	Included in <b>write_full</b> cross coverage bin that covers write cross full	Outputs are checked against reference model + <b>assert_overflow</b>
Read_Empty	Read from an empty FIFO and check for the underflow to be deasserted when rd_en is asserted	When the FIFO is empty constraint rd_en to be high most of the time and wr_en to be low most of the time	Included in <b>read_empty</b> cross coverage bin that covers read cross empty	Outputs are checked against reference model + <b>assert_underflow</b>
Reset	In reset cases check for the FIFO to be cleared and empty is asserted and full is deasserted	Constraint reset to be activated more frequently while randomizing the other inputs	Included in <b>full_to_empty</b> bin that covers the transition from a full FIFO to an empty FIFO (possible only if reset is activated)	Outputs are checked against reference model + <b>assert_count_rst, assert_wr_ptr_rst &amp; assert_rd_ptr_rst</b>
Simultaneous_Read_Write_Full	When the FIFO is full and reading and writing are taking place simultaneously, then overflow must remain low	When the FIFO is full, constraint wr_en and rd_en to be high simultaneously most of the time.	Included in <b>write_read_simit_full</b> cross coverage between <b>write_read_simit</b> and <b>full</b>	Outputs are checked against reference model + <b>assert_overflow</b>
Simultaneous_Read_Write_Empty	When the FIFO is empty and reading and writing are taking place simultaneously, then underflow must remain low	When the FIFO is empty, constraint wr_en and rd_en to be high simultaneously most of the time.	Included in <b>write_read_simit_empty</b> cross coverage between <b>write_read_simit</b> and <b>empty</b>	Outputs are checked against reference model + <b>assert_underflow</b>
Read_Write_Randomized	Perform random sequences of reads and writes to test the FIFO's robustness	Randomize inputs without constraints		Outputs are checked against reference model
Flags_Stability	When neither reading nor writing is taking place, flags must remain stable	Constraint rd_en and wr_en to be low most of the time	Included in <b>write_nor_read</b> that covers that write and read are deactivated simultaneously	Outputs are checked against reference model + <b>assert_full, assert_empty, assert_almostfull, assert_almostempty, assert_overflow, assert_underflow, assert_wr_ack</b>

# Code Snippets

## FIFO\_transaction\_pkg

```
1  package FIFO_transaction_pkg;
2
3      class FIFO_transaction;
4          parameter FIFO_WIDTH = 16;
5          parameter FIFO_DEPTH = 8;
6
7          rand bit [FIFO_WIDTH-1:0] data_in;
8          rand bit rst_n, wr_en, rd_en;
9          rand bit [FIFO_WIDTH-1:0] data_out;
10         rand bit wr_ack, overflow;
11         rand bit full, empty, almostfull, almostempty, underflow;
12
13         int WR_EN_ON_DIST, RD_EN_ON_DIST, RST_ON_DIST;
14
15         constraint Reset_Constraint {
16             rst_n dist {1:=(100-RST_ON_DIST), 0:=RST_ON_DIST};
17         }
18
19         constraint General_Constraints {
20             wr_en dist {1:=WR_EN_ON_DIST, 0:=(100-WR_EN_ON_DIST)};
21             rd_en dist {1:=RD_EN_ON_DIST, 0:=(100-RD_EN_ON_DIST)};
22             (full == 1) -> wr_en dist {1:=90, 0:=10};
23             (empty == 1) -> rd_en dist {1:=90, 0:=10};
24             unique {data_in};
25         }
26
27         constraint Simultaneous_Read_Write {
28             (wr_en && rd_en) dist {1:=90, 0:=10};
29             (full == 1) -> wr_en dist {1:=90, 0:=10};
30             (empty == 1) -> rd_en dist {1:=90, 0:=10};
31             unique {data_in};
32         }
33
34         function new(int RST_ON_DIST = 5, int WR_EN_ON_DIST = 70, int RD_EN_ON_DIST = 30);
35             this.RST_ON_DIST = RST_ON_DIST;
36             this.WR_EN_ON_DIST = WR_EN_ON_DIST;
37             this.RD_EN_ON_DIST = RD_EN_ON_DIST;
38         endfunction : new
39     endclass : FIFO_transaction
40
41 endpackage : FIFO_transaction_pkg
```

## FIFO\_coverage\_pkg

```
1 package FIFO_coverage_pkg;
2 import FIFO_transaction_pkg::*;
3
4 class FIFO_coverage;
5     FIFO_transaction F_cvgtxn = new();
6
7     covergroup FIFO_cvgt;
8         //Labeling coverpoints to be visible in the functional coverage report
9         write_cp: coverpoint F_cvgtxn.wr_en (bins wr_en[] = {[0:1]});
10        read_cp: coverpoint F_cvgtxn.rd_en (bins rd_en[] = {[0:1]});
11        full_cp: coverpoint F_cvgtxn.full (bins full[] = {[0:1]});
12        empty_cp: coverpoint F_cvgtxn.empty (bins empty[] = {[0:1]});
13        almostfull_cp: coverpoint F_cvgtxn.almostfull (bins almostfull[] = {[0:1]});
14        almostempty_cp: coverpoint F_cvgtxn.almostempty (bins almostempty[] = {[0:1]});
15        overflow_cp: coverpoint F_cvgtxn.overflow (bins overflow[] = {[0:1]});
16        underflow_cp: coverpoint F_cvgtxn.underflow (bins underflow[] = {[0:1]});
17        wr_ack_cp: coverpoint F_cvgtxn.wr_ack (bins wr_ack[] = {[0:1]});
18        not_to_empty_cp: coverpoint F_cvgtxn.empty (bins not_to_empty = {[0:1]});
19
20
21        write_seq_cp: coverpoint F_cvgtxn.wr_en (bins write_seq = (0->1["8"]->0));
22        read_seq_cp: coverpoint F_cvgtxn.rd_en (bins read_seq = (0->1["8"]->0));
23        full_to_not_cp: coverpoint F_cvgtxn.full (bins full_to_not = (1->0));
24        write_read_simlt_cp: cross write_cp, read_cp (bins write_read_simlt = binsof(write_cp.wr_en[1]) && binsof(read_cp.rd_en[1]); option.cross_auto_bin_max = 0;);
25        full_to_empty_cp: cross not_to_empty_cp, full_to_not_cp (bins full_to_empty = binsof(full_to_not_cp) && binsof(not_to_empty_cp); option.cross_auto_bin_max = 0;);
26        write_read_simlt_full_cp: cross write_read_simlt_cp, full_cp (bins write_read_simlt_full = binsof(write_read_simlt_cp) && binsof(full_cp.full[1]); option.cross_auto_bin_max = 0;);
27        write_read_simlt_empty_cp: cross write_read_simlt_cp, empty_cp (bins write_read_simlt_empty = binsof(write_read_simlt_cp) && binsof(empty_cp.empty[1]); option.cross_auto_bin_max = 0;);
28        write_nor_read_cp: cross write_cp, read_cp (bins write_read_simlt = binsof(write_cp.wr_en[0]) && binsof(read_cp.rd_en[0]); option.cross_auto_bin_max = 0;);
29
30        write_cross_states: cross write_cp, full_cp, empty_cp, almostfull_cp, almostempty_cp, overflow_cp, underflow_cp, wr_ack_cp {
31            bins write_full = binsof(write_cp) && binsof(full_cp);
32            bins write_empty = binsof(write_cp) && binsof(empty_cp);
33            bins write_almostfull = binsof(write_cp) && binsof(almostfull_cp);
34            bins write_almostempty = binsof(write_cp) && binsof(almostempty_cp);
35            bins write_overflow = binsof(write_cp) && binsof(overflow_cp);
36            bins write_underflow = binsof(write_cp) && binsof(underflow_cp);
37            bins write_wr_ack = binsof(write_cp) && binsof(wr_ack_cp);
38        }
39
40        read_cross_states: cross read_cp, full_cp, empty_cp, almostfull_cp, almostempty_cp, overflow_cp, underflow_cp, wr_ack_cp {
41            bins read_full = binsof(read_cp) && binsof(full_cp);
42            bins read_empty = binsof(read_cp) && binsof(empty_cp);
43            bins read_almostfull = binsof(read_cp) && binsof(almostfull_cp);
44            bins read_almostempty = binsof(read_cp) && binsof(almostempty_cp);
45            bins read_overflow = binsof(read_cp) && binsof(overflow_cp);
46            bins read_underflow = binsof(read_cp) && binsof(underflow_cp);
47            bins read_wr_ack = binsof(read_cp) && binsof(wr_ack_cp);
48        }
49    }
50
51    endgroup : FIFO_cvgt;
52
53    function void sample_data(FIFO_transaction F_txn);
54        F_cvgtxn = F_txn;
55        FIFO_cvgt.sample();
56    endfunction : sample_data
57
58    function new();
59        FIFO_cvgt = new();
60    endfunction : new
61 endclass : FIFO_coverage
62 endpackage : FIFO_coverage_pkg
```

## FIFO\_monitor

```
1 import shared_pkg::*;
2 import FIFO_transaction_pkg::*;
3 import FIFO_scoreboard_pkg::*;
4 import FIFO_coverage_pkg::*;
5
6 module FIFO_monitor (FIFO_if FIFOif);
7     FIFO_transaction FIFO_tr = new();
8     FIFO_scoreboard FIFO_sb = new();
9     FIFO_coverage FIFO_cvgt = new();
10
11    initial begin
12        forever @(negedge FIFOif.clk) begin
13            //Assigning inputs
14            FIFO_tr.data_in = FIFOif.data_in;
15            FIFO_tr.wr_en = FIFOif.wr_en;
16            FIFO_tr.rd_en = FIFOif.rd_en;
17            FIFO_tr.rst_n = FIFOif.rst_n;
18
19            //Assigning outputs
20            FIFO_tr.full = FIFOif.full;
21            FIFO_tr.empty = FIFOif.empty;
22            FIFO_tr.almostfull = FIFOif.almostfull;
23            FIFO_tr.almostempty = FIFOif.almostempty;
24            FIFO_tr.wr_ack = FIFOif.wr_ack;
25            FIFO_tr.overflow = FIFOif.overflow;
26            FIFO_tr.underflow = FIFOif.underflow;
27            FIFO_tr.data_out = FIFOif.data_out;
28
29            //Assigning Reference Outputs
30            FIFO_sb.full_ref = FIFOif.full_ref;
31            FIFO_sb.empty_ref = FIFOif.empty_ref;
32            FIFO_sb.almostfull_ref = FIFOif.almostfull_ref;
33            FIFO_sb.almostempty_ref = FIFOif.almostempty_ref;
34            FIFO_sb.wr_ack_ref = FIFOif.wr_ack_ref;
35            FIFO_sb.overflow_ref = FIFOif.overflow_ref;
36            FIFO_sb.underflow_ref = FIFOif.underflow_ref;
37            FIFO_sb.data_out_ref = FIFOif.data_out_ref;
38
39            fork
40                begin
41                    FIFO_cvgt.sample_data(FIFO_tr);
42                end
43                begin
44                    FIFO_sb.check_data(FIFO_tr);
45                end
46            join
47
48            if(test_finished) begin
49                $display("FIFO's test has been successfully finished\nerror_count = %0d\nincorrect_count = %0d", error_count, correct_count);
50                $stop;
51            end
52        end
53    end
54 end
55 endmodule : FIFO_monitor
```

# FIFO\_scoreboard\_pkg

```
1 package FIFO_scoreboard_pkg;
2 import FIFO_transaction_pkg::*;
3 import shared_pkg::*;
4
5 class FIFO_scoreboard;
6     FIFO_transaction F_sb_txn = new();
7
8     parameter FIFO_WIDTH = 16;
9     parameter FIFO_DEPTH = 8;
10
11     bit [FIFO_WIDTH-1:0] data_out_ref;
12     bit wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;
13
14     bit [FIFO_WIDTH-1:0] FIFO_model[$]; //golden model for the FIFO
15
16     task check_data(FIFO_transaction F_txn);
17         F_sb_txn = F_txn;
18
19         if (F_sb_txn.data_out == data_out_ref)
20             correct_count++;
21         else begin
22             $error("Mismatch: data_out (%0d) != data_out_ref (%0d)", F_sb_txn.data_out, data_out_ref);
23             error_count++;
24         end
25
26         if (F_sb_txn.full == full_ref)
27             correct_count++;
28         else begin
29             $error("Mismatch: full (%0d) != full_ref (%0d)", F_sb_txn.full, full_ref);
30             error_count++;
31         end
32
33         if (F_sb_txn.empty == empty_ref)
34             correct_count++;
35         else begin
36             $error("Mismatch: empty (%0d) != empty_ref (%0d)", F_sb_txn.empty, empty_ref);
37             error_count++;
38         end
39
40         if (F_sb_txn.almostfull == almostfull_ref)
41             correct_count++;
42         else begin
43             $error("Mismatch: almostfull (%0d) != almostfull_ref (%0d)", F_sb_txn.almostfull, almostfull_ref);
44             error_count++;
45         end
46
47         if (F_sb_txn.almostempty == almostempty_ref)
48             correct_count++;
49         else begin
50             $error("Mismatch: almostempty (%0d) != almostempty_ref (%0d)", F_sb_txn.almostempty, almostempty_ref);
51             error_count++;
52         end
53
54         if (F_sb_txn.overflow == overflow_ref)
55             correct_count++;
56         else begin
57             $error("Mismatch: overflow (%0d) != overflow_ref (%0d)", F_sb_txn.overflow, overflow_ref);
58             error_count++;
59         end
60
61         if (F_sb_txn.underflow == underflow_ref)
62             correct_count++;
63         else begin
64             $error("Mismatch: underflow (%0d) != underflow_ref (%0d)", F_sb_txn.underflow, underflow_ref);
65             error_count++;
66         end
67
68         if (F_sb_txn.wr_ack == wr_ack_ref)
69             correct_count++;
70         else begin
71             $error("Mismatch: wr_ack (%0d) != wr_ack_ref (%0d)", F_sb_txn.wr_ack, wr_ack_ref);
72             error_count++;
73         end
74     endtask : check_data
75
76 endclass : FIFO_scoreboard
77
78 endpackage : FIFO_scoreboard_pkg
```

## FIFO\_ref (reference model)

```
1  module FIFO_ref(FIFO_if,REF FIFOif);
2      logic [FIFOif.FIFO_WIDTH-1:0] mem[$];
3      bit wr_ack_next, wr_ack_next1;
4
5      always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
6          if(~FIFOif.rst_n) begin
7              mem.delete();
8              FIFOif.overflow_ref <= 0;
9              FIFOif.wr_ack_ref <= 0;
10         end
11         else begin
12             if(FIFOif.wr_en) begin
13                 if(!FIFOif.full_ref) begin
14                     mem.push_front(FIFOif.data_in);
15                     FIFOif.wr_ack_ref <= 1;
16                     FIFOif.overflow_ref <= 0;
17                 end else begin
18                     FIFOif.wr_ack_ref <= 0;
19                     FIFOif.overflow_ref <= 1;
20                 end
21             end else begin
22                 FIFOif.wr_ack_ref <= 0;
23                 FIFOif.overflow_ref <= 0;
24             end
25         end
26     end
27
28     always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
29         if(~FIFOif.rst_n) begin
30             mem.delete();
31             FIFOif.data_out_ref <= 0;
32             FIFOif.underflow_ref <= 0;
33         end
34         else begin
35             if(FIFOif.rd_en) begin
36                 if(!FIFOif.empty_ref) begin
37                     FIFOif.data_out_ref <= mem.pop_back();
38                     FIFOif.underflow_ref <= 0;
39                 end else begin
40                     FIFOif.underflow_ref <= 1;
41                 end
42             end else
43                 FIFOif.underflow_ref <= 0;
44         end
45     end
46
47     always_comb begin
48         case(mem.size())
49             0:
50                 begin
51                     FIFOif.full_ref = 0;
52                     FIFOif.almostfull_ref = 0;
53                     FIFOif.empty_ref = 1;
54                     FIFOif.almostempty_ref = 0;
55                 end
56             1:
57                 begin
58                     FIFOif.full_ref = 0;
59                     FIFOif.almostfull_ref = 0;
60                     FIFOif.empty_ref = 0;
61                     FIFOif.almostempty_ref = 1;
62                 end
63             FIFOif.FIFO_DEPTH-1:
64                 begin
65                     FIFOif.full_ref = 0;
66                     FIFOif.almostfull_ref = 1;
67                     FIFOif.empty_ref = 0;
68                     FIFOif.almostempty_ref = 0;
69                 end
70             FIFOif.FIFO_DEPTH:
71                 begin
72                     FIFOif.full_ref = 1;
73                     FIFOif.almostfull_ref = 0;
74                     FIFOif.empty_ref = 0;
75                     FIFOif.almostempty_ref = 0;
76                 end
77             default:
78                 begin
79                     FIFOif.full_ref = 0;
80                     FIFOif.almostfull_ref = 0;
81                     FIFOif.empty_ref = 0;
82                     FIFOif.almostempty_ref = 0;
83                 end
84         endcase // mem.size()
85     end
86
87 endmodule
```

## FIFO\_tb

```
1  import shared_pkg::*;
2  import FIFO_transaction_pkg::*;
3
4  module FIFO_tb(FIFO_if.TEST FIFOif);
5      FIFO_transaction FIFO_tr = new();
6
7      //Sequence Items
8      FIFO_transaction tr_wr = new(5, 95, 5);    //Only_Write
9      FIFO_transaction tr_rd = new(5, 5, 95);    //Only_Read
10     FIFO_transaction tr_sm = new(5, 90, 90);    //Simultaneous_Read_Write
11     FIFO_transaction tr_wf = new(5, 90, 10);    //Write_Full
12     FIFO_transaction tr_re = new(5, 10, 90);    //Read_Empty
13     FIFO_transaction tr_rs = new(50, 70, 30);   //Reset
14     FIFO_transaction tr_sf = new(5, 100, 100);  //Simultaneous_Read_Write_Full
15
16
17     initial begin
18         //Customizing the constraints for each test case
19         prepare_items();
20
21         //Initialize the design
22         reset();
23
24         //Only_Write
25         repeat(500) begin
26             randomize_then_assign(tr_wr);
27             @(negedge FIFOif.clk);
28         end
29
30         //Only_Read
31         repeat(500) begin
32             randomize_then_assign(tr_rd);
33             @(negedge FIFOif.clk);
34         end
35
36         //Concurrent_Write_Read
37         repeat(100) begin
38             repeat(10) begin
39                 randomize_then_assign(tr_wr);
40                 @(negedge FIFOif.clk);
41             end
42             repeat(10) begin
43                 randomize_then_assign(tr_rd);
44                 @(negedge FIFOif.clk);
45             end
46         end
47
48         //Simultaneous_Read_Write
49         repeat(500) begin
50             randomize_then_assign(tr_sm);
51             @(negedge FIFOif.clk);
52         end
53
54         //Reset
55         repeat(500) begin
56             randomize_then_assign(tr_rs);
57             @(negedge FIFOif.clk);
58         end
59
60         test_finished = 1;
61     end
62
63     task reset();
64         FIFOif.rst_n = 0;
65         @(negedge FIFOif.clk);
66         FIFOif.rst_n = 1;
67     endtask : reset
```



```

68
69     function void randomize_then_assign(FIFO_transaction tr);
70         FIFO_tr = tr;
71         assert(FIFO_tr.randomize());
72         //Assigning inputs
73         FIFOif.data_in = FIFO_tr.data_in;
74         FIFOif.wr_en = FIFO_tr.wr_en;
75         FIFOif.rd_en = FIFO_tr.rd_en;
76         FIFOif.rst_n = FIFO_tr.rst_n;
77     endfunction : randomize_then_assign
78
79     task fill_FIFO();
80         FIFOif.rst_n = 1;
81         FIFOif.wr_en = 1;
82         FIFOif.rd_en = 0;
83         repeat(10) begin
84             FIFOif.data_in = $random();
85             @(negedge FIFOif.clk);
86         end
87     endtask : fill_FIFO
88
89     task empty_FIFO();
90         FIFOif.rst_n = 1;
91         FIFOif.wr_en = 0;
92         FIFOif.rd_en = 1;
93         repeat(10) begin
94             FIFOif.data_in = $random();
95             @(negedge FIFOif.clk);
96         end
97     endtask : empty_FIFO
98
99     task prepare_items();
100         tr_wr.constraint_mode(0);
101         tr_wr.Reset_Constraint.constraint_mode(1);
102         tr_wr.General_Constraints.constraint_mode(1);
103
104         tr_rd.constraint_mode(0);
105         tr_rd.Reset_Constraint.constraint_mode(1);
106         tr_rd.General_Constraints.constraint_mode(1);
107
108         tr_sm.constraint_mode(0);
109         tr_sm.Reset_Constraint.constraint_mode(1);
110         tr_sm.Simultaneous_Read_Write.constraint_mode(1);
111
112         tr_rs.constraint_mode(0);
113         tr_rs.Reset_Constraint.constraint_mode(1);
114         tr_rs.General_Constraints.constraint_mode(1);
115     endtask : prepare_items
116
117 endmodule : FIFO_tb

```

## FIFO\_if

```

1  interface FIFO_if #(
2      parameter FIFO_WIDTH = 16,
3      parameter FIFO_DEPTH = 8)(input bit clk);
4
5      Logic [FIFO_WIDTH-1:0] data_in;
6      Logic rst_n, wr_en, rd_en;
7      Logic [FIFO_WIDTH-1:0] data_out;
8      Logic wr_ack, overflow;
9      Logic full, empty, almostfull, almostempty, underflow;
10
11     Logic [FIFO_WIDTH-1:0] data_out_ref;
12     Logic wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;
13
14     modport DUT (
15         input data_in, clk, rst_n, wr_en, rd_en,
16         output data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow
17     );
18
19     modport REF (
20         input data_in, clk, rst_n, wr_en, rd_en,
21         output data_out_ref, wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref
22     );
23
24     modport TEST (
25         output data_in, rst_n, wr_en, rd_en,
26         input clk, data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow
27     );
28
29     modport MONITOR (input data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull, almostempty, wr_ack, overflow, underflow, data_out);
30 endinterface : FIFO_if

```



## FIFO\_top

```
1  module FIFO_top ();
2      bit clk;
3
4      initial begin
5          forever #10 clk = ~clk;
6      end
7
8      FIFO_if FIFOif(clk);
9      FIFO DUT(FIFOif);
10     FIFO_ref REF(FIFOif);
11     FIFO_tb TEST(FIFOif);
12     FIFO_monitor MONITOR(FIFOif);
13 endmodule : FIFO_top
```

## run\_FIFO (Do File)

```
1  vlib work
2  vlog -f src_files.list -mfcu +define+SIM +cover
3  vsim -voptargs=+acc work.FIFO_top -cover
4  add wave -position 1 -color red sim:/FIFO_top/FIFOif/clk
5  add wave -position 2 sim:/FIFO_top/FIFOif/rst_n
6  add wave -position 3 sim:/FIFO_top/FIFOif/wr_en
7  add wave -position 4 sim:/FIFO_top/FIFOif/rd_en
8  add wave -position 5 sim:/FIFO_top/FIFOif/data_in
9  add wave -position 6 -color cyan sim:/FIFO_top/FIFOif/data_out
10 add wave -position 7 -color gold sim:/FIFO_top/FIFOif/data_out_ref
11 add wave -position 8 -color cyan sim:/FIFO_top/FIFOif/wr_ack
12 add wave -position 9 -color gold sim:/FIFO_top/FIFOif/wr_ack_ref
13 add wave -position 10 -color cyan sim:/FIFO_top/FIFOif/full
14 add wave -position 11 -color gold sim:/FIFO_top/FIFOif/full_ref
15 add wave -position 12 -color cyan sim:/FIFO_top/FIFOif/empty
16 add wave -position 13 -color gold sim:/FIFO_top/FIFOif/empty_ref
17 add wave -position 14 -color cyan sim:/FIFO_top/FIFOif/almostfull
18 add wave -position 15 -color gold sim:/FIFO_top/FIFOif/almostfull_ref
19 add wave -position 16 -color cyan sim:/FIFO_top/FIFOif/almostempty
20 add wave -position 17 -color gold sim:/FIFO_top/FIFOif/almostempty_ref
21 add wave -position 18 -color cyan sim:/FIFO_top/FIFOif/overflow
22 add wave -position 19 -color gold sim:/FIFO_top/FIFOif/overflow_ref
23 add wave -position 20 -color cyan sim:/FIFO_top/FIFOif/underflow
24 add wave -position 21 -color gold sim:/FIFO_top/FIFOif/underflow_ref
25 add wave -position 22 -color blue sim:/FIFO_top/DUT/mem
26 add wave -position 23 -color blue sim:/FIFO_top/DUT/count
27 add wave -position 24 -color blue sim:/FIFO_top/DUT/wr_ptr
28 add wave -position 25 -color blue sim:/FIFO_top/DUT/rd_ptr
29 add wave -position 26 sim:/FIFO_top/DUT/assert_full
30 add wave -position 27 sim:/FIFO_top/DUT/assert_empty
31 add wave -position 28 sim:/FIFO_top/DUT/assert_almostfull
32 add wave -position 29 sim:/FIFO_top/DUT/assert_almostempty
33 add wave -position 30 sim:/FIFO_top/DUT/assert_overflow
34 add wave -position 31 sim:/FIFO_top/DUT/assert_underflow
35 add wave -position 32 sim:/FIFO_top/DUT/assert_wr_ack
36 add wave -position 33 sim:/FIFO_top/DUT/assert_count_incr
37 add wave -position 34 sim:/FIFO_top/DUT/assert_count_decr
38 add wave -position 35 sim:/FIFO_top/DUT/assert_wr_ptr_incr
39 add wave -position 36 sim:/FIFO_top/DUT/assert_rd_ptr_incr
40 add wave -position 37 sim:/FIFO_top/DUT/assert_count_rst
41 add wave -position 38 sim:/FIFO_top/DUT/assert_wr_ptr_rst
42 add wave -position 39 sim:/FIFO_top/DUT/assert_rd_ptr_rst
43 .vcop Action toggleleafnames
44 coverage save FIFO_tb.ucdb -onexit -du FIFO
45 run -all
46 coverage report -detail -cvlg -directive -comments -output fcover_report.txt {}
47 quit -sim
48 vcover report FIFO_tb.ucdb -details -annotate -all -output coverage_rpt.txt
```

## src\_files.list

```
1  shared_pkg.sv
2  FIFO_transaction.sv
3  FIFO_coverage.sv
4  FIFO_scoreboard.sv
5  FIFO_if.sv
6  FIFO_monitor.sv
7  FIFO_tb.sv
8  FIFO_top.sv
9  FIFO.sv
10 FIFO_ref.sv
```

# Detected Bugs

## 1. Extended count bus by 1-bit to allow it to reach FIFO\_DEPTH

### Before:

```
22 reg [max_fifo_addr:0] count;
```

### After:

```
15 reg [max_fifo_addr+1:0] count;
```

## 2. Added this line to deassert overflow incase of writing

### Before:

```
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
```

### After:

```
21 else if (FIFOif.wr_en && count < FIFOif.FIFO_DEPTH) begin
22     mem[wr_ptr] <= FIFOif.data_in;
23     FIFOif.wr_ack <= 1;
24     wr_ptr <= wr_ptr + 1;
25     FIFOif.overflow <= 0;
26 end
```

## 3. Added this line to deassert underflow incase of reading

### Before:

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

### After:

```
40 else if (FIFOif.rd_en && count != 0) begin
41     FIFOif.data_out <= mem[rd_ptr];
42     rd_ptr <= rd_ptr + 1;
43     FIFOif.underflow <= 0;
44 end
```

## 4. Removed this line since underflow is a sequential output not combinational

### Before:

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

### After:

```
73 //assign FIFOif.underflow = (FIFOif.empty && FIFOif.rd_en)? 1 : 0;
```

## 5. Added the sequential implementation of the underflow

### Before:

```
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
```

### After:

```
36  always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
37      if (!FIFOif.rst_n) begin
38          rd_ptr <= 0;
39      end
40      else if (FIFOif.rd_en && count != 0) begin
41          FIFOif.data_out <= mem[rd_ptr];
42          rd_ptr <= rd_ptr + 1;
43          FIFOif.underflow <= 0;
44      end
45      else begin
46          if (FIFOif.empty & FIFOif.rd_en)
47              FIFOif.underflow <= 1;
48          else
49              FIFOif.underflow <= 0;
50      end
51  end
```

## 6. Modified this line as FIFO is almostfull when count reaches FIFO\_DEPTH-1

### Before:

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

### After:

```
74  assign FIFOif.almostfull = (count == FIFOif.FIFO_DEPTH-1)? 1 : 0;
```

## 7. Added this line as data\_out must asynchronously be resetted (sequential)

### Before:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        rd_ptr <= 0;
```

### After:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        FIFOif.data_out <= 0;
        rd_ptr <= 0;
        FIFOif.underflow <= 0;
```

## 8. Added this line as overflow must asynchronously be resetted (sequential)

### Before:

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

### After:

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

## 9. Added this line as underflow must asynchronously be resetted (sequential)

### Before:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        rd_ptr <= 0;
```

### After:

```
always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
    if (!FIFOif.rst_n) begin
        FIFOif.data_out <= 0;
        rd_ptr <= 0;
        FIFOif.underflow <= 0;
```

## 10. Added this line as wr\_ack must asynchronously be resetted (sequential)

### Before:

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

### After:

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

# DUT after fixing bugs and adding assertions

```
8 module FIFO(FIFO_if,DUT FIFOif);
9
10 localparam max_fifo_addr = $log2(FIFOif.FIFO_DEPTH);
11
12 reg [FIFOif.FIFO_WIDTH-1:0] mem [FIFOif.FIFO_DEPTH-1:0];
13
14 reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
15 reg [max_fifo_addr-1:0] count; //modified_code(1) /*to allow count to reach FIFO_DEPTH*/
16
17 always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
18     if (!FIFOif.rst_n) begin
19         wr_ptr <= 0;
20         FIFOif.overflow <= 0; //added_code (1) /*overflow must asynchronously be resetted (sequential)*/
21         FIFOif.wr_ack <= 0; //added_code (2) /*overflow must asynchronously be resetted (sequential)*/
22     end
23     else if (FIFOif.wr_en && count < FIFOif.FIFO_DEPTH) begin
24         mem[wr_ptr] <= FIFOif.data_in;
25         FIFOif.wr_ack <= 1;
26         wr_ptr <= wr_ptr + 1;
27         FIFOif.overflow <= 0; //added_code (3) /*overflow should be deasserted in the case of writing*/
28     end
29     else begin
30         FIFOif.wr_ack <= 0;
31         if (FIFOif.full & FIFOif.wr_en)
32             FIFOif.overflow <= 1;
33         else
34             FIFOif.overflow <= 0;
35     end
36 end
37
38 always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
39     if (!FIFOif.rst_n) begin
40         FIFOif.data_out <= 0; //added_code (4) /*data_out must asynchronously be resetted (sequential)*/
41         rd_ptr <= 0;
42         FIFOif.underflow <= 0; //added_code (5) /*underflow must asynchronously be resetted (sequential)*/
43     end
44     else if (FIFOif.rd_en && count != 0) begin
45         FIFOif.data_out <= mem[rd_ptr];
46         rd_ptr <= rd_ptr + 1;
47         FIFOif.underflow <= 0; //added_code (6) /*underflow should be deasserted in the case of reading*/
48     end
49     else begin
50         if (FIFOif.empty & FIFOif.rd_en) //added_code (7) begin /*added the sequential implementation of underflow*/
51             FIFOif.underflow <= 1; //*****
52         else //*****
53             FIFOif.underflow <= 0; //*****
54         //added_code (7) end
55     end
56 end
57
58 always @(posedge FIFOif.clk or negedge FIFOif.rst_n) begin
59     if (!FIFOif.rst_n) begin
60         count <= 0;
61     end
62     else begin
63         if (((FIFOif.wr_en, FIFOif.rd_en) == 2'b10) && !FIFOif.full)
64             count <= count + 1;
65         else if (((FIFOif.wr_en, FIFOif.rd_en) == 2'b01) && !FIFOif.empty)
66             count <= count - 1;
67         else if ((FIFOif.wr_en, FIFOif.rd_en) == 2'b11) begin //added_code (8) begin /*added the part that handles simultaneous read-write*/
68             case ((FIFOif.full, FIFOif.empty)) //*****
69                 2'b01: count <= count + 1; //*****
70                 2'b10: count <= count - 1; //*****
71             endcase //added_code (8) end
72         end
73     end
74 end
75
76 assign FIFOif.full = (count == FIFOif.FIFO_DEPTH)? 1 : 0;
77 assign FIFOif.empty = (count == 0)? 1 : 0;
78 //assign FIFOif.underflow = (FIFOif.empty && FIFOif.rd_en)? 1 : 0; //removed_code /*underflow is sequential and not combinational*/
79 assign FIFOif.almostfull = (count == FIFOif.FIFO_DEPTH-1)? 1 : 0; //modified_code (2) /*FIFO is almostfull when count reaches FIFO_DEPTH-1 not FIFO_DEPTH-2 as 0 is not included*/
80 assign FIFOif.almostempty = (count == 1)? 1 : 0;
81
82 `ifdef SIM
83     //Assertions
84
85     //Assertions to the combinational flags
86     always_comb begin
87         if(count == FIFOif.FIFO_DEPTH) begin
88             assert_full: assert final (FIFOif.full == 1) else $error("Mismatch: full_tb (%0d) != full_ref (1)", FIFOif.full);
89             cover_full: cover final (FIFOif.full == 1);
90         end
91
92         if(count == 0) begin
93             assert_empty: assert final (FIFOif.empty == 1) else $error("Mismatch: empty_tb (%0d) != empty_ref (1)", FIFOif.empty);
94             cover_empty: cover final (FIFOif.empty == 1);
95         end
96
97         if(count == FIFOif.FIFO_DEPTH-1) begin
98             assert_almostfull: assert final (FIFOif.almostfull == 1) else $error("Mismatch: almostfull_tb (%0d) != almostfull_ref (1)", FIFOif.almostfull);
99             cover_almostfull: cover final (FIFOif.almostfull == 1);
100         end
101
102         if(count == 1) begin
103             assert_almostempty: assert final (FIFOif.almostempty == 1) else $error("Mismatch: almostempty_tb (%0d) != almostempty_ref (1)", FIFOif.almostempty);
104             cover_almostempty: cover final (FIFOif.almostempty == 1);
105         end
106     end
107
108     //Assertions to the sequential flags
109     assert_overflow: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && FIFOif.full) |>= (FIFOif.overflow == 1));
110     cover_overflow: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && FIFOif.full) |>= (FIFOif.overflow == 1));
111
112     assert_underflow: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.rd_en && FIFOif.empty) |>= (FIFOif.underflow == 1));
113     cover_underflow: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.rd_en && FIFOif.empty) |>= (FIFOif.underflow == 1));
114
115     assert_wr_ack: assert property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && !FIFOif.full) |>= (FIFOif.wr_ack == 1));
116     cover_wr_ack: cover property (@(posedge FIFOif.clk) disable iff (!FIFOif.rst_n) (FIFOif.wr_en && !FIFOif.full) |>= (FIFOif.wr_ack == 1));
117
118     property count_plus;
119     @(posedge FIFOif.clk)
120     disable iff (!FIFOif.rst_n)
121     (FIFOif.wr_en && !FIFOif.full && !FIFOif.rd_en) || (FIFOif.wr_en && !FIFOif.full && FIFOif.rd_en && FIFOif.empty) |>= (count == $past(count) + 1);
122 endproperty
123
124     property count_minus;
125     @(posedge FIFOif.clk)
126     disable iff (!FIFOif.rst_n)
127     (FIFOif.rd_en && !FIFOif.empty && !FIFOif.wr_en) || (FIFOif.rd_en && !FIFOif.empty && FIFOif.wr_en && FIFOif.full) |>= (count == $past(count) - 1);
128 endproperty
129
130     property wr_ptr_plus;
131     @(posedge FIFOif.clk)
132     disable iff (!FIFOif.rst_n)
133     (FIFOif.wr_en && !FIFOif.full) |>= (wr_ptr == $past(wr_ptr) + 1) || (wr_ptr == 0);
134 endproperty
135
136     property rd_ptr_plus;
137     @(posedge FIFOif.clk)
138     disable iff (!FIFOif.rst_n)
139     (FIFOif.rd_en && !FIFOif.empty) |>= (rd_ptr == $past(rd_ptr) + 1) || (rd_ptr == 0);
140 endproperty
141
142     property write_pointer_stable;
143     @(posedge FIFOif.clk) disable iff(!FIFOif.rst_n)
144     FIFOif.full && FIFOif.wr_en && !FIFOif.rd_en |>= $stable(wr_ptr);
145 endproperty
146
147     property read_pointer_stable;
148     @(posedge FIFOif.clk) disable iff(!FIFOif.rst_n)
149     FIFOif.empty && !FIFOif.rd_en && FIFOif.wr_en |>= $stable(rd_ptr);
150 endproperty
151
152     //Assertions to the counters
153     assert_count_incr: assert property (count_plus);
154     cover_count_incr: cover property (count_plus);
155
156     assert_count_decr: assert property (count_minus);
157     cover_count_decr: cover property (count_minus);
158
159     assert_wr_ptr_incr: assert property (wr_ptr_plus);
160     cover_wr_ptr_incr: cover property (wr_ptr_plus);
161
162     assert_rd_ptr_incr: assert property (rd_ptr_plus);
163     cover_rd_ptr_incr: cover property (rd_ptr_plus);
164
165     assert_write_pointer_stable: assert property (write_pointer_stable);
166     cover_write_pointer_stable: cover property (write_pointer_stable);
167
168     assert_read_pointer_stable: assert property (read_pointer_stable);
169     cover_read_pointer_stable: cover property (read_pointer_stable);
170
171     always_comb begin
172         if(!FIFOif.rst_n) begin
173             assert_count_rst: assert final (count == 0);
174             cover_count_rst: cover final (count == 0);
175
176             assert_wr_ptr_rst: assert final (wr_ptr == 0);
177             cover_wr_ptr_rst: cover final (wr_ptr == 0);
178
179             assert_rd_ptr_rst: assert final (rd_ptr == 0);
180             cover_rd_ptr_rst: cover final (rd_ptr == 0);
181         end
182     end
183
184 `endif
185 endmodule
```

# QuestaSim Snippets

## Wave colour guidelines

1. Cyan for the DUT outputs
2. Gold for the reference outputs
3. Red for the clock
4. Green for the inputs
5. Blue for the internal signals

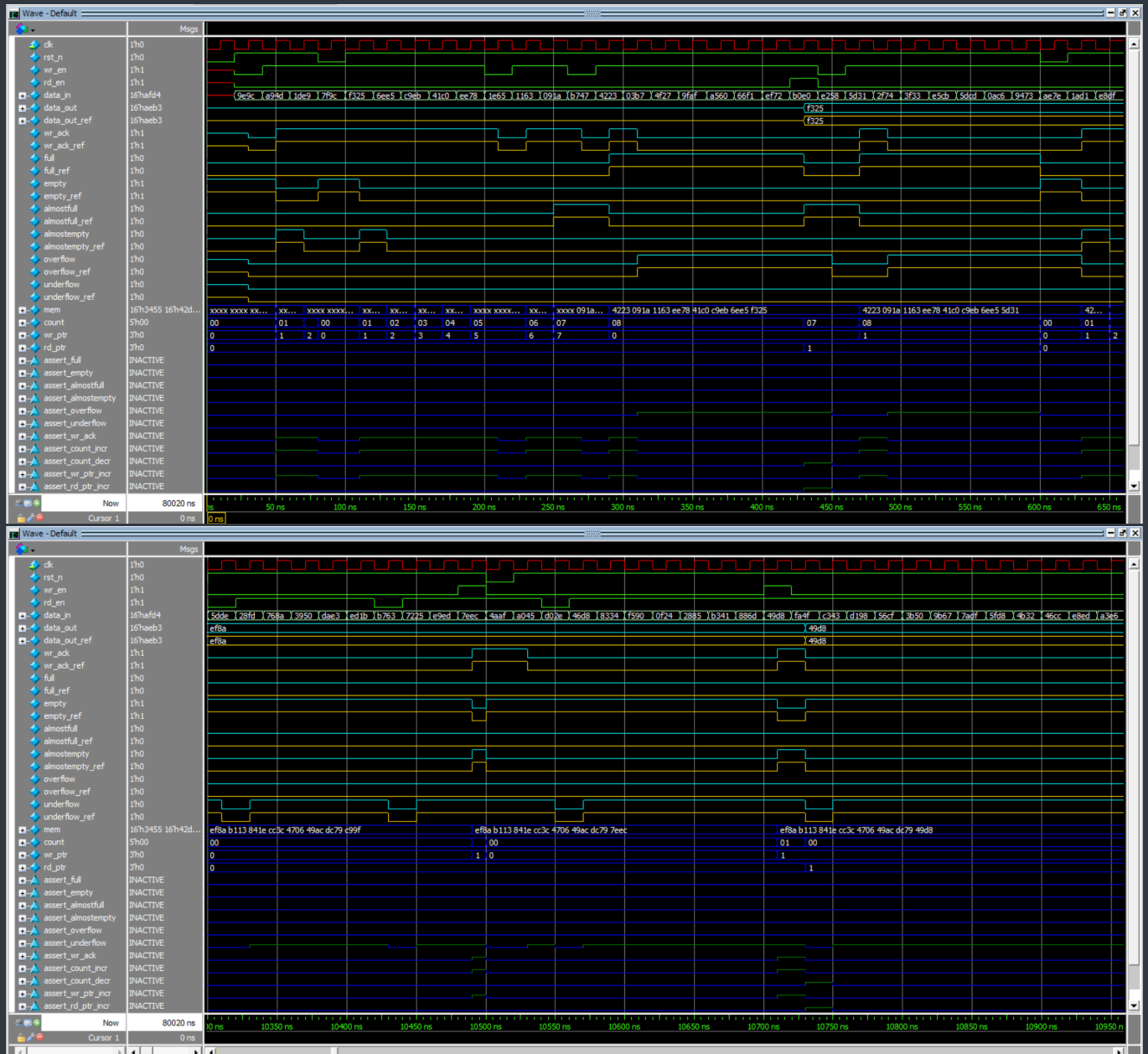
## Snippets

### Transcript

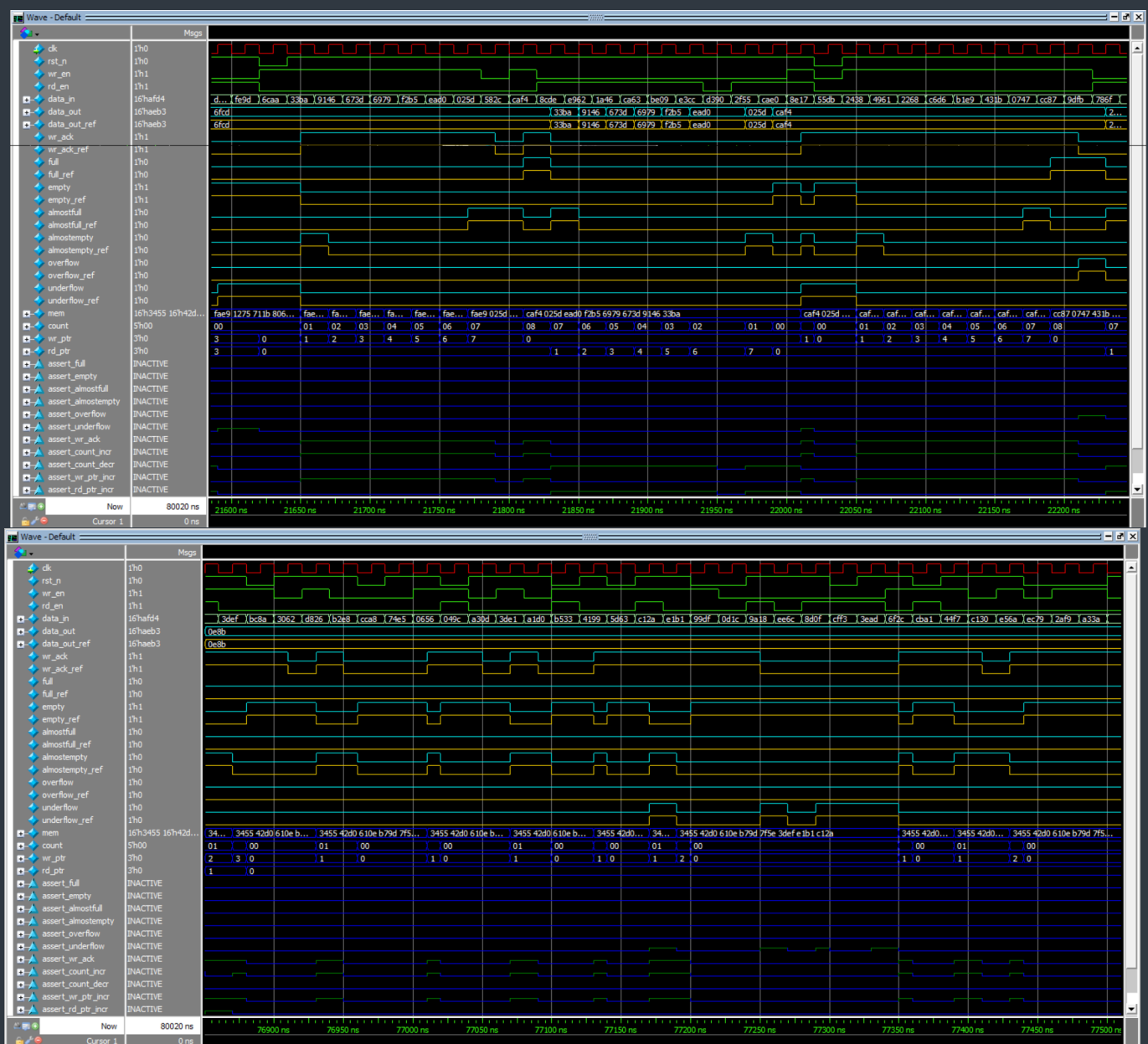
```
# FIFO's test has been successfully finished
# error_count = 0
# correct_count = 32008
# ** Note: $stop      : FIFO_monitor.sv(51)
#   Time: 80020 ns  Iteration: 1  Instance: /FIFO_top/MONITOR
# Break in Module FIFO_monitor at FIFO_monitor.sv line 51
```

VSIM 3>

### Waveform







## Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/FIFO_coverage_pkg/FIFO_coverage		100.00%							
TYPEB FIFO_cvg_gp		100.00%	100	100.00...					auto(0)
CVP FIFO_cvg_gp::write_cp		100.00%	100	100.00...					
CVP FIFO_cvg_gp::read_cp		100.00%	100	100.00...					
CVP FIFO_cvg_gp::full_cp		100.00%	100	100.00...					
CVP FIFO_cvg_gp::empty_cp		100.00%	100	100.00...					
CVP FIFO_cvg_gp::almostfull_cp		100.00%	100	100.00...					
CVP FIFO_cvg_gp::almostempty_cp		100.00%	100	100.00...					
CVP FIFO_cvg_gp::overflow_cp		100.00%	100	100.00...					
CVP FIFO_cvg_gp::underflow_cp		100.00%	100	100.00...					
CVP FIFO_cvg_gp::wr_ack_cp		100.00%	100	100.00...					
CVP FIFO_cvg_gp::not_to_empty_cp		100.00%	100	100.00...					
CVP FIFO_cvg_gp::write_seq_cp		100.00%	100	100.00...					
CVP FIFO_cvg_gp::read_seq_cp		100.00%	100	100.00...					
CVP FIFO_cvg_gp::empty_to_not_cp		100.00%	100	100.00...					
CVP FIFO_cvg_gp::full_to_not_cp		100.00%	100	100.00...					
CROSS FIFO_cvg_gp::write_read_simlt_c...		100.00%	100	100.00...					
CROSS FIFO_cvg_gp::full_to_empty_cp		100.00%	100	100.00...					
CROSS FIFO_cvg_gp::write_read_simlt_f...		100.00%	100	100.00...					
CROSS FIFO_cvg_gp::write_read_simlt_e...		100.00%	100	100.00...					
CROSS FIFO_cvg_gp::write_nor_read_cp		100.00%	100	100.00...					
CROSS FIFO_cvg_gp::write_cross_states		100.00%	100	100.00...					
CROSS FIFO_cvg_gp::read_cross_states		100.00%	100	100.00...					
INST FIFO_coverage_pkg::FIFO_cover...		100.00%	100	100.00...					0

## Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
/FIFO_top/DUT/assert_full	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert (FIFOOf.full)	
/FIFO_top/DUT/assert_empty	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert (FIFOOf.empty)	
/FIFO_top/DUT/assert_almostfull	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert (FIFOOf.almostfull)	
/FIFO_top/DUT/assert_almostempty	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert (FIFOOf.almostempty)	
/FIFO_top/DUT/assert_overflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@(posedge FIFOOf.dk) disa...	
/FIFO_top/DUT/assert_underflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@(posedge FIFOOf.dk) disa...	
/FIFO_top/DUT/assert_wr_ack	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@(posedge FIFOOf.dk) disa...	
/FIFO_top/DUT/assert_count_inc	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@(posedge FIFOOf.dk) disa...	
/FIFO_top/DUT/assert_count_dec	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@(posedge FIFOOf.dk) disa...	
/FIFO_top/DUT/assert_wr_ptr_inc	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@(posedge FIFOOf.dk) disa...	
/FIFO_top/DUT/assert_rd_ptr_inc	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert (@(posedge FIFOOf.dk) disa...	
/FIFO_top/DUT/assert_count_rst	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert (count==0)	
/FIFO_top/DUT/assert_wr_ptr_rst	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert (wr_ptr==0)	
/FIFO_top/DUT/assert_rd_ptr_rst	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert (rd_ptr==0)	
/FIFO_top/TEST/andonize_then_assign/immed_71	Immediate	SVA	on	0	1	-	-	-	-	0	off	assert (andonize(...))	

# Cover Directives

Cover Directives														
Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
▲ /FIFO_top/DUT/cover_full	SVA	✓	Off	91	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_empty	SVA	✓	Off	350	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_almostfull	SVA	✓	Off	159	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_almostempty	SVA	✓	Off	420	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_overflow	SVA	✓	Off	403	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_underflow	SVA	✓	Off	789	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_wir_ack	SVA	✓	Off	1455	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_count_inc	SVA	✓	Off	1036	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_count_dec	SVA	✓	Off	546	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_wir_ptr_inc	SVA	✓	Off	1455	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_rd_ptr_inc	SVA	✓	Off	965	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_write_pointer_stable	SVA	✓	Off	384	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_read_pointer_stable	SVA	✓	Off	698	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_count_rst	SVA	✓	Off	311	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_wir_ptr_rst	SVA	✓	Off	311	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0
▲ /FIFO_top/DUT/cover_rd_ptr_rst	SVA	✓	Off	311	1	Unli...	1	100%	<div></div>	✓	0	0	0 ns	0

# Coverage Reports

## Code Coverage Report

**Note: Conditional Coverage didn't reach 100% as the the two conditions (!FIFOif.full && FIFOif.wr\_en) and (!FIFOif.empty && FIFOif.rd\_en) would never happen if else was entered; because in case one of them happened else statement wouldn't be entered. As a result, Total Coverage By Instance dropped to 96.51%.**

```
186 Condition Coverage:
187   Enabled Coverage          Bins   Covered   Misses   Coverage
188   -----
189   Conditions                24     22        2    91.66%
190
191 =====Condition Details=====
192
193 Condition Coverage for instance /\FIFO_top#DUT --
194
195   File FIFO.sv
196   -----Focused Condition View-----
197   Line      21 Item    1 (FIFOif.wr_en && (count < FIFOif.FIFO_DEPTH))
198   Condition totals: 2 of 2 input terms covered = 100.00%
199
200   Input Term   Covered   Reason for no coverage   Hint
201   -----
202   FIFOif.wr_en      Y
203   (count < FIFOif.FIFO_DEPTH)  Y
204
205   Rows:      Hits   FEC Target      Non-masking condition(s)
206   -----
207   Row 1:      1   FIFOif.wr_en_0      -
208   Row 2:      1   FIFOif.wr_en_1      (count < FIFOif.FIFO_DEPTH)
209   Row 3:      1   (count < FIFOif.FIFO_DEPTH)_0  FIFOif.wr_en
210   Row 4:      1   (count < FIFOif.FIFO_DEPTH)_1  FIFOif.wr_en
211
212   -----Focused Condition View-----
213   Line      29 Item    1 (FIFOif.full & FIFOif.wr_en)
214   Condition totals: 1 of 2 input terms covered = 50.00%
215
216   Input Term   Covered   Reason for no coverage   Hint
217   -----
218   FIFOif.full      N   '_0' not hit           Hit '_0'
219   FIFOif.wr_en      Y
220
221   Rows:      Hits   FEC Target      Non-masking condition(s)
222   -----
223   Row 1:  ***0***  FIFOif.full_0      FIFOif.wr_en
224   Row 2:      1   FIFOif.full_1      FIFOif.wr_en
225   Row 3:      1   FIFOif.wr_en_0      FIFOif.full
226   Row 4:      1   FIFOif.wr_en_1      FIFOif.full
227
228   -----Focused Condition View-----
229   Line      40 Item    1 (FIFOif.rd_en && (count != 0))
230   Condition totals: 2 of 2 input terms covered = 100.00%
231
232   Input Term   Covered   Reason for no coverage   Hint
233   -----
234   FIFOif.rd_en      Y
235   (count != 0)      Y
236
237   Rows:      Hits   FEC Target      Non-masking condition(s)
238   -----
239   Row 1:      1   FIFOif.rd_en_0      -
240   Row 2:      1   FIFOif.rd_en_1      (count != 0)
241   Row 3:      1   (count != 0)_0      FIFOif.rd_en
242   Row 4:      1   (count != 0)_1      FIFOif.rd_en
243
244   -----Focused Condition View-----
245   Line      46 Item    1 (FIFOif.empty & FIFOif.rd_en)
246   Condition totals: 1 of 2 input terms covered = 50.00%
247
248   Input Term   Covered   Reason for no coverage   Hint
249   -----
250   FIFOif.empty      N   '_0' not hit           Hit '_0'
251   FIFOif.rd_en      Y
252
253   Rows:      Hits   FEC Target      Non-masking condition(s)
254   -----
255   Row 1:  ***0***  FIFOif.empty_0      FIFOif.rd_en
256   Row 2:      1   FIFOif.empty_1      FIFOif.rd_en
257   Row 3:      1   FIFOif.rd_en_0      FIFOif.empty
258   Row 4:      1   FIFOif.rd_en_1      FIFOif.empty
```

```
21 else if (FIFOif.wr_en && count < FIFOif.FIFO_DEPTH) begin
```

```
22   mem[wr_ptr] <= FIFOif.data_in;
23   FIFOif.wr_ack <= 1;
24   wr_ptr <= wr_ptr + 1;
25   FIFOif.overflow <= 0;
```

```
26 end
```

```
27 else begin
```

```
28   FIFOif.wr_ack <= 0;
```

```
29   if (FIFOif.full & FIFOif.wr_en)
```

```
30     FIFOif.overflow <= 1;
```

```
31   else
```

```
32     FIFOif.overflow <= 0;
```

```
33 end
```

```
40 else if (FIFOif.rd_en && count != 0) begin
```

```
41   FIFOif.data_out <= mem[rd_ptr];
42   rd_ptr <= rd_ptr + 1;
43   FIFOif.underflow <= 0;
```

```
44 end
```

```
45 else begin
```

```
46   if (FIFOif.empty & FIFOif.rd_en)
```

```
47     FIFOif.underflow <= 1;
```

```
48   else
```

```
49     FIFOif.underflow <= 0;
```

```
50 end
```

**Note: Toggle coverage didn't reach 100% as count is 4-bits while it resets to zero when reaching 8**

```
680 Toggle Coverage:
681   Enabled Coverage           Bins      Hits      Misses  Coverage
682   -----
683   Toggles                    22       20         2    90.90%
684
685   =====Toggle Details=====
686
687 Toggle Coverage for instance /\FIFO_top#DUT  --
688
689                                     Node      1H->0L      0L->1H  "Coverage"
690                                     -----
691                                     count[4]         0         0      0.00
692                                     count[3-0]        1         1     100.00
693                                     rd_ptr[2-0]        1         1     100.00
694                                     wr_ptr[2-0]        1         1     100.00
695
696 Total Node Count      =      11
697 Toggled Node Count   =      10
698 Untoggled Node Count =       1
699
700 Toggle Coverage      =     90.90% (20 of 22 bins)
```

## Functional Coverage Report

```
412 missing/total bins:      0      1      -
413 % Hit:                   100.00%  100
414 bin empty_to_not        349      1      -  Covered
415 Coverpoint full_to_not_cp 100.00%  100      -  Covered
416 covered/total bins:      1      1      -
417 missing/total bins:      0      1      -
418 % Hit:                   100.00%  100
419 bin full_to_not         91      1      -  Covered
420 Cross write_read_simlt_cp 100.00%  100      -  Covered
421 covered/total bins:      1      1      -
422 missing/total bins:      0      1      -
423 % Hit:                   100.00%  100
424 Auto, Default and User Defined Bins:
425   bin write_read_simlt    4001      1      -  Covered
426 Cross full_to_empty_cp    100.00%  100      -  Covered
427 covered/total bins:      1      1      -
428 missing/total bins:      0      1      -
429 % Hit:                   100.00%  100
430 Auto, Default and User Defined Bins:
431   bin full_to_empty       17      1      -  Covered
432 Cross write_read_simlt_full_cp 100.00%  100      -  Covered
433 covered/total bins:      1      1      -
434 missing/total bins:      0      1      -
435 % Hit:                   100.00%  100
436 Auto, Default and User Defined Bins:
437   bin write_read_simlt_full 4001      1      -  Covered
438 Cross write_read_simlt_empty_cp 100.00%  100      -  Covered
439 covered/total bins:      1      1      -
440 missing/total bins:      0      1      -
441 % Hit:                   100.00%  100
442 Auto, Default and User Defined Bins:
443   bin write_read_simlt_empty 4001      1      -  Covered
444 Cross write_nor_read_cp    100.00%  100      -  Covered
445 covered/total bins:      1      1      -
446 missing/total bins:      0      1      -
447 % Hit:                   100.00%  100
448 Auto, Default and User Defined Bins:
449   bin write_read_simlt    4001      1      -  Covered
450 Cross write_cross_states    100.00%  100      -  Covered
451 covered/total bins:      7      7      -
452 missing/total bins:      0      7      -
453 % Hit:                   100.00%  100
454 Auto, Default and User Defined Bins:
455   bin write_full          4001      1      -  Covered
456   bin write_empty         4001      1      -  Covered
457   bin write_almostfull    4001      1      -  Covered
458   bin write_almostempty   4001      1      -  Covered
459   bin write_overflow       4001      1      -  Covered
460   bin write_underflow     4001      1      -  Covered
461   bin write_wr_ack        4001      1      -  Covered
462 Cross read_cross_states    100.00%  100      -  Covered
463 covered/total bins:      7      7      -
464 missing/total bins:      0      7      -
465 % Hit:                   100.00%  100
466 Auto, Default and User Defined Bins:
467   bin read_full           4001      1      -  Covered
468   bin read_empty          4001      1      -  Covered
469   bin read_almostfull     4001      1      -  Covered
470   bin read_almostempty    4001      1      -  Covered
471   bin read_overflow       4001      1      -  Covered
472   bin read_underflow      4001      1      -  Covered
473   bin read_wr_ack         4001      1      -  Covered
474
475 TOTAL COVERGROUP COVERAGE: 100.00% COVERGROUP TYPES: 1
476
477 Total Coverage By Instance (filtered view): 100.00%
```

# Sequential Domain Coverage Report

492	Cross read_cross_states	100.00%	100	-	Covered
493	covered/total bins:	7	7	-	
494	missing/total bins:	0	7	-	
495	% Hit:	100.00%	100	-	
496	Auto, Default and User Defined Bins:				
497	bin read_full	4001	1	-	Covered
498	bin read_empty	4001	1	-	Covered
499	bin read_almostfull	4001	1	-	Covered
500	bin read_almostempty	4001	1	-	Covered
501	bin read_overflow	4001	1	-	Covered
502	bin read_underflow	4001	1	-	Covered
503	bin read_wr_ack	4001	1	-	Covered

505 TOTAL COVERGROUP COVERAGE: 100.00% COVERGROUP TYPES: 1

## 507 DIRECTIVE COVERAGE:

509 Name	Design Unit	Design UnitType	Lang	File(Line)	Hits	Status
512 /FIFO_top/DUT/cover_full	FIFO	Verilog	SVA	FIFO.sv(89)	91	Covered
513 /FIFO_top/DUT/cover_empty	FIFO	Verilog	SVA	FIFO.sv(94)	350	Covered
514 /FIFO_top/DUT/cover_almostfull	FIFO	Verilog	SVA	FIFO.sv(99)	159	Covered
515 /FIFO_top/DUT/cover_almostempty	FIFO	Verilog	SVA	FIFO.sv(104)	420	Covered
516 /FIFO_top/DUT/cover_overflow	FIFO	Verilog	SVA	FIFO.sv(111)	403	Covered
517 /FIFO_top/DUT/cover_underflow	FIFO	Verilog	SVA	FIFO.sv(114)	789	Covered
518 /FIFO_top/DUT/cover_wr_ack	FIFO	Verilog	SVA	FIFO.sv(117)	1455	Covered
519 /FIFO_top/DUT/cover_count_incr	FIFO	Verilog	SVA	FIFO.sv(155)	1036	Covered
520 /FIFO_top/DUT/cover_count_decr	FIFO	Verilog	SVA	FIFO.sv(158)	546	Covered
521 /FIFO_top/DUT/cover_wr_ptr_incr	FIFO	Verilog	SVA	FIFO.sv(161)	1455	Covered
522 /FIFO_top/DUT/cover_rd_ptr_incr	FIFO	Verilog	SVA	FIFO.sv(164)	965	Covered
523 /FIFO_top/DUT/cover_write_pointer_stable	FIFO	Verilog	SVA	FIFO.sv(167)	384	Covered
524 /FIFO_top/DUT/cover_read_pointer_stable	FIFO	Verilog	SVA	FIFO.sv(170)	698	Covered
525 /FIFO_top/DUT/cover_count_rst	FIFO	Verilog	SVA	FIFO.sv(175)	311	Covered
526 /FIFO_top/DUT/cover_wr_ptr_rst	FIFO	Verilog	SVA	FIFO.sv(178)	311	Covered
527 /FIFO_top/DUT/cover_rd_ptr_rst	FIFO	Verilog	SVA	FIFO.sv(181)	311	Covered

529 TOTAL DIRECTIVE COVERAGE: 100.00% COVERS: 16

531 Total Coverage By Instance (filtered view): 100.00%