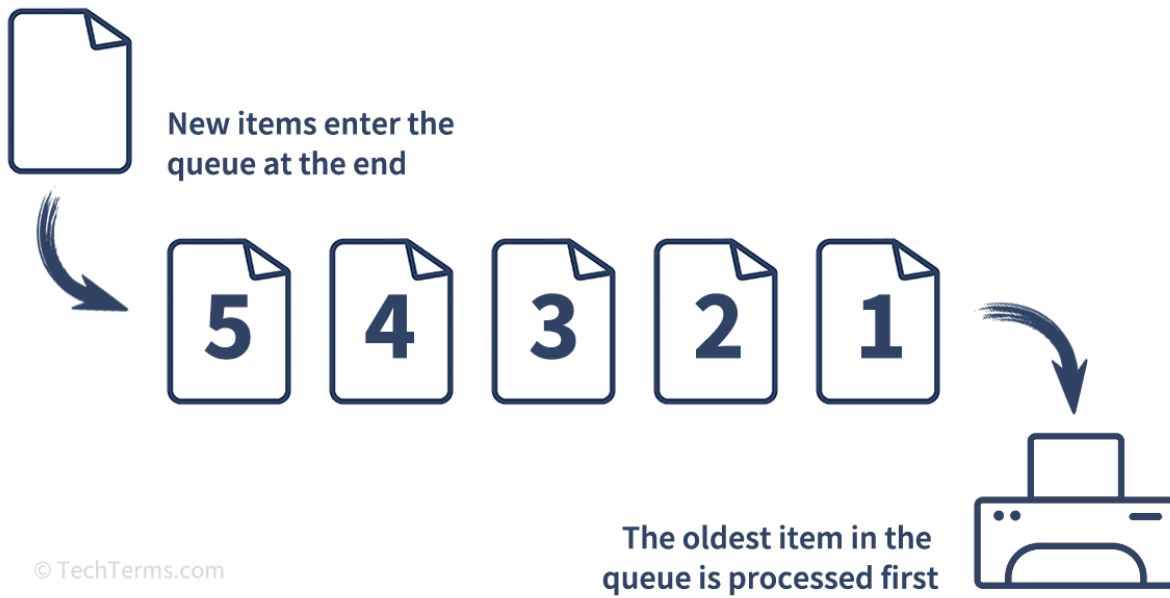


SV Project - Synchronous FIFO
Supervisor: Eng. Kareem Waseem



Mostafa Kamal Mohamed

Table of contents

I.	Definition of FIFO
II.	Verification Plan
III.	RTL Bugs Report
IV.	Design and SVA codes snippets
V.	Do File
VI.	Questa snippets
VII.	Coverage Report

I. Definition of FIFO

Stands for "First In, First Out."

FIFO is a method for organizing, processing or retrieving data or other objects in a queue. In a FIFO system, the data that has been waiting the longest gets processed first whenever there is an opening. New objects are added to the back of the queue and must wait their turn as the system processes each object in order.

The FIFO model is one of the most basic ways to process data. It does not allow one object to jump the line or weigh the priority of multiple items when choosing what to process next — everything waits its turn without exception. The opposite of the FIFO model is the LIFO model, or last-in-first-out, where the newest entry is processed first.

Many computer queues operate using a FIFO model. For example, a network printer in a busy office will use a FIFO queue to schedule print jobs as they come in, even if your document is only two pages and the job right before yours is a hundred.

Computers also typically use FIFO scheduling when pulling data from an array or buffer. Disk write scheduling, process scheduling, and message systems also often use a FIFO model to handle requests in order without consideration for high or low priority.

II. Verification Plan

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	When the reset is asserted, the internal pointers, counter and overflow, underflow will reset	Directed at the start of the sim, then randomized with constraint that drives reset to be off most of the sim time	-	immediate assertion to check the reset functionality
FIFO_2	When wr_en and rd_en are high and the fifo is empty, the priority is to write to fifo	Randomization under constraint on wr_en to be high 70% of the time and rd_en to be high 30% Of the time	coverpoints wr_en and rd_e n and cross cover with empty and full signals	concurrent assertion to check count is increased
FIFO_3	When wr_en and rd_en are high and the fifo is full, the priority is to read from fifo	Randomization under constraint on wr_en to be high 70% of the time and rd_en to be high 30% Of the time	coverpoints wr_en and rd_en and cross cover with empty and full signals	concurrent assertion to check count is decreased
FIFO_4	When wr_en is high and rd_en is low and the fifo is not full, write operation is done	Randomization under constraint on wr_en to be high 70% of the time	cross cover between wr_en and full signal	concurrent assertion to check count is increased, self check using reference model
FIFO_5	When wr_en is low and rd_en is high and the fifo is not empty, read operation is done	Randomization under constraint on wr_en to be high 30% of the time	cross cover between rd_en and empty signal	concurrent assertion to check count is decreased, self check using reference model
FIFO_6	When count is equal to depth - 1		cross cover between almostfull and wr_en and rd_en	immediate assertion to check almostfull is high, self check using reference model
FIFO_7	When count is equal to 1		cross cover between almostempty and wr_en and rd_en	immediate assertion to check almostempty is high, self check using reference model
FIFO_8	When wr_en is high and fifo is not full. wr_ack is high		cross cover between wr_ack and wr_en and rd_en	concurrent assertion to check wr_ack is high, self check using reference model
FIFO_9	When wr_en is high and fifo is full, overflow is high		cross cover between overflow and wr_en and rd_en	concurrent assertion to check overflow is high, self check using reference model
FIFO_10	When rd_en is high and fifo is empty, underflow is high		cross cover between underflow and wr_en and rd_en	concurrent assertion to check underflow is high, self check using reference model
FIFO_11	When count = depth, fifo is full		cross cover between full and wr_en and rd_en	immediate assertion to check full is high, self check using reference model
FIFO_12	When count = 0, fifo is empty		cross cover between empty and wr_en and rd_en	immediate assertion to check empty is high, self check using reference model
FIFO_13	data_in	Randomized during the simulation		

III. RTL Bugs Report

1.

Original design

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

Modified design

```
if (!f_if.rst_n) begin
    wr_ptr <= 0;
    f_if.overflow <= 0; // reset overflow signal
    f_if.wr_ack <= 0; // reset wr_ack signal
end
```

2.

Original design

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

Modified design

```
if (!f_if.rst_n) begin
    rd_ptr <= 0;
    f_if.underflow <= 0; // reset underflow signal
end
```

3.

Original design

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

Modified design

```
else begin
    if (f_if.empty && f_if.rd_en) // make underflow signal sequential
        f_if.underflow <= 1;
    else
        f_if.underflow <= 0;
end
```

4.

Original design

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

Modified design

```
assign f_if.almostfull = (count == f_if.FIFO_DEPTH-1) ? 1 : 0; // modify the almostfull signal to match design specs
```

5.

Original design

```
else begin
    if ( ({wr_en, rd_en} == 2'b10) && !full)
        count <= count + 1;
    else if ( ({wr_en, rd_en} == 2'b01) && !empty)
        count <= count - 1;
end
```

Modified design

```
else begin
    if ({f_if.wr_en, f_if.rd_en} == 2'b10 && !f_if.full)
        count <= count + 1;
    else if ({f_if.wr_en, f_if.rd_en} == 2'b01 && !f_if.empty)
        count <= count - 1;
    else if ({f_if.wr_en, f_if.rd_en} == 2'b11 && f_if.full) // add unhandled case
        count <= count - 1;
    else if ({f_if.wr_en, f_if.rd_en} == 2'b11 && f_if.empty) // add unhandled case
        count <= count + 1;
end
```

IV. Design and SVA Codes snippets

1. Assertions

```
`ifdef SIM
// reset
always_comb begin
    if(!f_if.rst_n)
        reset: assert final(!wr_ptr && !rd_ptr && !count);
end
//full
always_comb begin
    if(f_if.rst_n && (count== f_if.FIFO_DEPTH))
        one_full: assert final(f_if.full);
    //almostfull
    if(f_if.rst_n && (count== f_if.FIFO_DEPTH-1))
        two_almostfull: assert final(f_if.almostfull);
    //empty
    if(f_if.rst_n && (count== 0))
        three_empty: assert final(f_if.empty);
    //almostempty
    if(f_if.rst_n && (count== 1))
        four_almostempty: assert final(f_if.almostempty);
end
//overflow
property five;
@posedge f_if.clk disable iff(!f_if.rst_n) (f_if.full && f_if.wr_en) |>=> f_if.overflow;
endproperty
overflow_flag_assert: assert property(five);
overflow_flag_cover: cover property(five);
//underflow
property six;
@posedge f_if.clk disable iff(!f_if.rst_n) (f_if.empty && f_if.rd_en) |>=> f_if.underflow;
endproperty
underflow_flag_assert: assert property(six);
underflow_flag_cover: cover property(six);
`endif
```

```

//wr_ack
property seven;
@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en && count < f_if.FIFO_DEPTH) | => f_if.wr_ack;
endproperty
wr_ack_flag_assert: assert property(seven);
wr_ack_flag_cover: cover property(seven);
//write
property eight;
@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en && !f_if.rd_en && !f_if.full) | => $past(count+ 1'b1);
endproperty
write_assert: assert property(eight);
write_cover: cover property(eight);
//read
property nine;
@(posedge f_if.clk) disable iff(!f_if.rst_n) (!f_if.wr_en && f_if.rd_en && !f_if.empty) | => (count+ 1'b1);
endproperty
read_assert: assert property(nine);
read_cover: cover property(nine);
//write priority
property ten;
@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en && f_if.rd_en && f_if.full) | => (count+ 1'b1);
endproperty
write_pri_assert: assert property(ten);
write_pri_cover: cover property(ten);
//read priority
property eleven;
@(posedge f_if.clk) disable iff(!f_if.rst_n) (f_if.wr_en && f_if.rd_en && f_if.empty) | => $past(count+ 1'b1);
endproperty
read_pri_assert: assert property(eleven);
read_pri_cover: cover property(eleven);
`endif

```

2. FIFO_top

```

module FIFO_top();
bit clk;

// clock generation
initial begin
|   clk = 0;
forever begin
|   #5 clk = ~clk;
end
end
//instantiate interface and design
FIFO_if f_if(clk);
FIFO_DUT(f_if);
FIFO_tb TEST(f_if);
FIFO_monitor MONITOR(f_if);
endmodule

```

3. FIFO_transaction (constraints)

```

int RD_EN_ON_DIST = 30;
int WR_EN_ON_DIST = 70;

constraint rst_dist{rst_n dist {0:=1, 1:=99}}; //rst
constraint write_enable_dist { wr_en dist {0 := 100 - WR_EN_ON_DIST, 1 := WR_EN_ON_DIST}}; } //write
constraint read_enable_dist { rd_en dist {0 := 100 - RD_EN_ON_DIST, 1 := RD_EN_ON_DIST}}; } //read

```

4. FIFO_monitor

```
fork
begin
    fifo_coverage.sample_data(fifo_txn);
end

begin
    @(posedge f_if.clk);
    #2;
    fifo_scoreboard.check_data(fifo_txn);
end
join
if (test_finished) begin
    $display("Test finished!");
    $display("Correct Count: %0d", correct_count);
    $display("Error Count: %0d", error_count);
    $stop;
end
```

5.

6. FIFO_scoreboard (check and reference functions)

```
function void check_data(input FIFO_transaction fifo_txn);
    bit [6:0] ref_flags, dut_flags;

    reference_model(fifo_txn);

    ref_flags = {full_ref, empty_ref, almostfull_ref, almostempty_ref, overflow_ref, underflow_ref, wr_ack_ref};
    dut_flags = {fifo_txn.full, fifo_txn.empty, fifo_txn.almostfull, fifo_txn.almostempty, fifo_txn.overflow, fifo_txn.underflow, fifo_txn.wr_ack};

    if (fifo_txn.data_out != data_out_ref) begin
        error_count++;
        $display("time:%0t Error: Data out mismatch. Expected: %0h, Got: %0h", $time, data_out_ref, fifo_txn.data_out);
    end else begin
        correct_count++;
    end

    if (ref_flags != dut_flags) begin
        error_count++;
        $display("time:%0t Error: Flags mismatch. Expected: %b, Got: %b", $time, ref_flags, dut_flags);
    end else begin
        correct_count++;
    end
endfunction
```

```
//reference model
function void reference_model(input FIFO_transaction fifo_txn);
    fork
        begin // write
            if (!fifo_txn.rst_n) begin
                wr_ack_ref = 0;
                overflow_ref = 0;
                full_ref = 0;
                almostfull_ref = 0;
                mem_queue.delete();
            end else if (fifo_txn.wr_en && count < FIFO_DEPTH) begin
                wr_ack_ref = 1;
                mem_queue.push_back(fifo_txn.data_in);
            end else begin
                wr_ack_ref = 0;
                overflow_ref = (full_ref && fifo_txn.wr_en) ? 1 : 0;
            end
        end

        begin //read
            if (!fifo_txn.rst_n) begin
                empty_ref = 1;
                underflow_ref = 0;
                almostempty_ref = 0;
            end else if (fifo_txn.rd_en && count != 0) begin
                data_out_ref = mem_queue.pop_front();
            end else begin
                underflow_ref = (empty_ref && fifo_txn.rd_en) ? 1 : 0;
            end
        end
    join
```



```

if (!fifo_txn.rst_n) begin // count
    count = 0;
end else if (fifo_txn.wr_en && !fifo_txn.rd_en && !full_ref) begin
    count = count + 1;
end else if (!fifo_txn.wr_en && fifo_txn.rd_en && !empty_ref) begin
    count = count - 1;
end else if (fifo_txn.wr_en && fifo_txn.rd_en && full_ref) begin
    count = count - 1;
end else if (fifo_txn.wr_en && fifo_txn.rd_en && empty_ref) begin
    count = count + 1;
end
//flags
full_ref = (count == FIFO_DEPTH) ? 1 : 0;
empty_ref = (count == 0) ? 1 : 0;
almostfull_ref = (count == FIFO_DEPTH - 1) ? 1 : 0;
almostempty_ref = (count == 1) ? 1 : 0;
endfunction

```

7. FIFO_coverage (covergroup)

```

covergroup FIFO_cvgr;
// Coverpoints
wr_en_cp:      coverpoint F_cvg_txn.wr_en;
rd_en_cp:      coverpoint F_cvg_txn.rd_en;
full_cp:       coverpoint F_cvg_txn.full;
empty_cp:      coverpoint F_cvg_txn.empty;
almostfull_cp: coverpoint F_cvg_txn.almostfull;
almostempty_cp: coverpoint F_cvg_txn.almostempty;
overflow_cp:   coverpoint F_cvg_txn.overflow;
underflow_cp:  coverpoint F_cvg_txn.underflow;
wr_ack_cp:     coverpoint F_cvg_txn.wr_ack;
// Cross coverage
wr_full:       cross wr_en_cp, full_cp;
wr_empty:      cross wr_en_cp, empty_cp;
wr_almostfull: cross wr_en_cp, almostfull_cp;
wr_almostempty: cross wr_en_cp, almostempty_cp;
wr_overflow:   cross wr_en_cp, overflow_cp{
    ignore_bins wr_en0_overflow1 = !binsof(wr_en_cp) intersect{1} && binsof(overflow_cp) intersect{1};
}
wr_underflow:  cross wr_en_cp, underflow_cp;
wr_wr_ack:     cross wr_en_cp, wr_ack_cp{
    ignore_bins wr_en0_wr_ack1 = !binsof(wr_en_cp) intersect{1} && binsof(wr_ack_cp) intersect{1};
}

rd_full:       cross rd_en_cp, full_cp{
    ignore_bins rd_en1_full1 = binsof(rd_en_cp) intersect{1} && binsof(full_cp) intersect{1};
}
rd_empty:      cross rd_en_cp, empty_cp;
rd_almostfull: cross rd_en_cp, almostfull_cp;
rd_almostempty: cross rd_en_cp, almostempty_cp;
rd_overflow:   cross rd_en_cp, overflow_cp;
rd_underflow:  cross rd_en_cp, underflow_cp{
    ignore_bins rd_en0_underflow1 = !binsof(rd_en_cp) intersect{1} && binsof(underflow_cp) intersect{1};
}
rd_wr_ack:     cross rd_en_cp, wr_ack_cp;
endgroup

```

8. FIFO_test

```

initial begin
    f_if.data_in = 0; f_if.wr_en = 0; f_if.rd_en = 0;
    assert_reset();
    repeat(100000) begin
        assert(fifo_txn.randomize());
        f_if.rst_n = fifo_txn.rst_n;
        f_if.wr_en = fifo_txn.wr_en;
        f_if.rd_en = fifo_txn.rd_en;
        f_if.data_in = fifo_txn.data_in;
        @(negedge f_if.clk);
    end
    test_finished = 1; // signal to be asserted when test finished
end

```

V. Do File

```
vlib work
vlog -f src_fifo_files.txt -mfcu +define+SIM +cover
vsim -voptargs+=+acc work.FIFO_top -classdebug -uvmcontrol=all -cover
add wave /FIFO_top/f_if/*
coverage save FIFO_top.ucdb -onexit
run -all
```

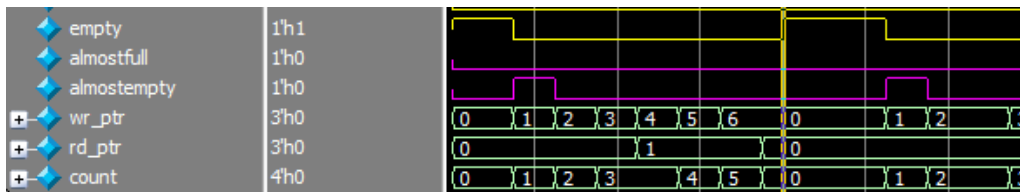
VI. Questa snippets

1. Report of results

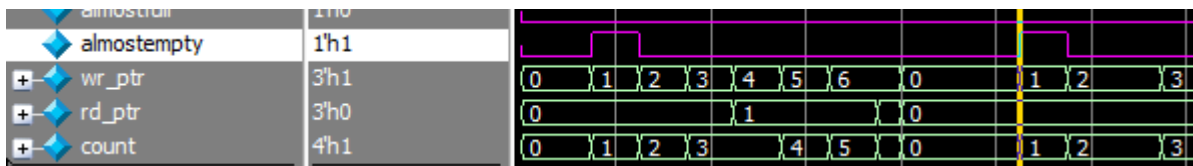
```
# Test finished!
# Correct Count: 200002
# Error Count: 0
# ** Note: $stop      : FIFO_monitor.sv(42)
#   Time: 1000017 ns  Iteration: 0  Instance: /FIFO_top/MONITOR
# Break in Module FIFO_monitor at FIFO_monitor.sv line 42
```

2. Waveform snippets

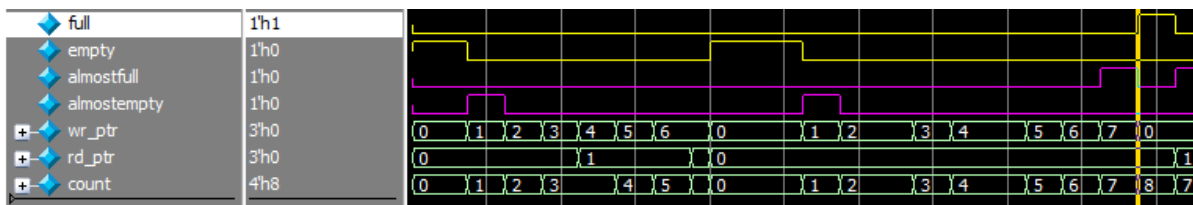
- Empty signal is high when count == zero



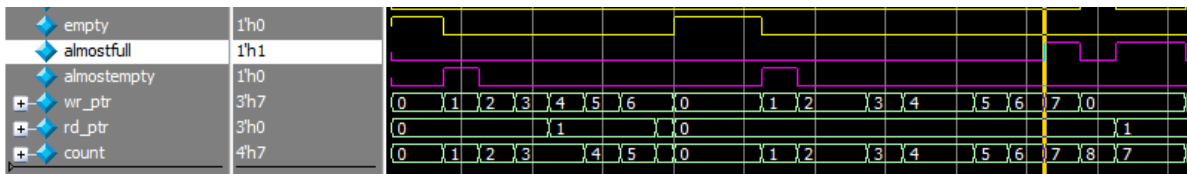
- Almostempty is high when count == 1



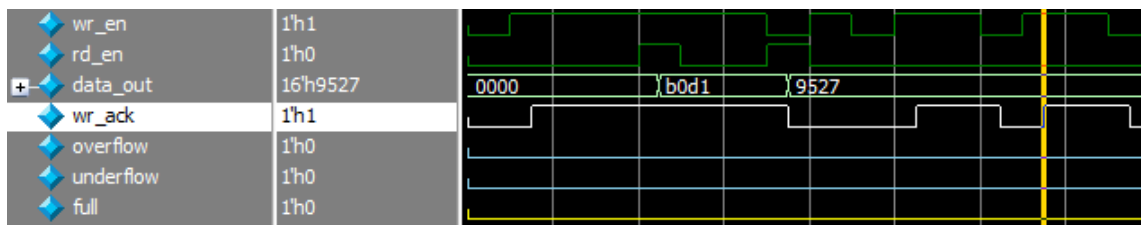
- Full signal is high when count == 8



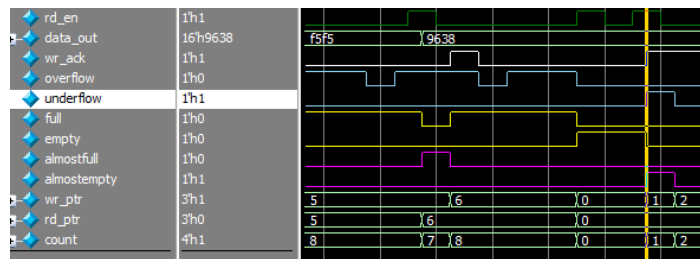
- Almostfull signal is high when count == 7



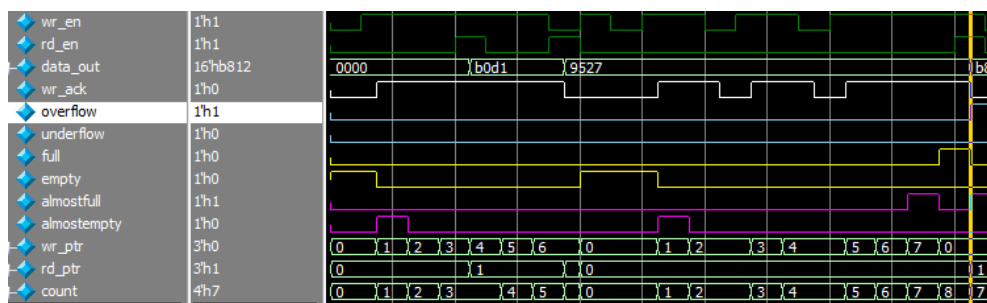
- When wr_en is high and not full wr_ack is high



- When rd_en is high and fifo is empty underflow signal is high



- When wr_en is high and full overflow is high



3. Assertions passed correctly

Assertions					
Name	Assertion Type	Language	Enable	Failure Count	Pass Count
+/FIFO_top/DUT/reset	Immediate	SVA	on	0	1
+/FIFO_top/DUT/one_full	Immediate	SVA	on	0	1
+/FIFO_top/DUT/two_almostfull	Immediate	SVA	on	0	1
+/FIFO_top/DUT/three_empty	Immediate	SVA	on	0	1
+/FIFO_top/DUT/four_almostempty	Immediate	SVA	on	0	1
+/FIFO_top/DUT/overflow_flag_assert	Concurrent	SVA	on	0	1
+/FIFO_top/DUT/underflow_flag_assert	Concurrent	SVA	on	0	1
+/FIFO_top/DUT/wr_ack_flag_assert	Concurrent	SVA	on	0	1
+/FIFO_top/DUT/write_assert	Concurrent	SVA	on	0	1
+/FIFO_top/DUT/read_assert	Concurrent	SVA	on	0	1
+/FIFO_top/DUT/write_pri_assert	Concurrent	SVA	on	0	1
+/FIFO_top/DUT/read_pri_assert	Concurrent	SVA	on	0	1
+/FIFO_top/TEST/#ublk#182146786#9/immed__10	Immediate	SVA	on	0	1

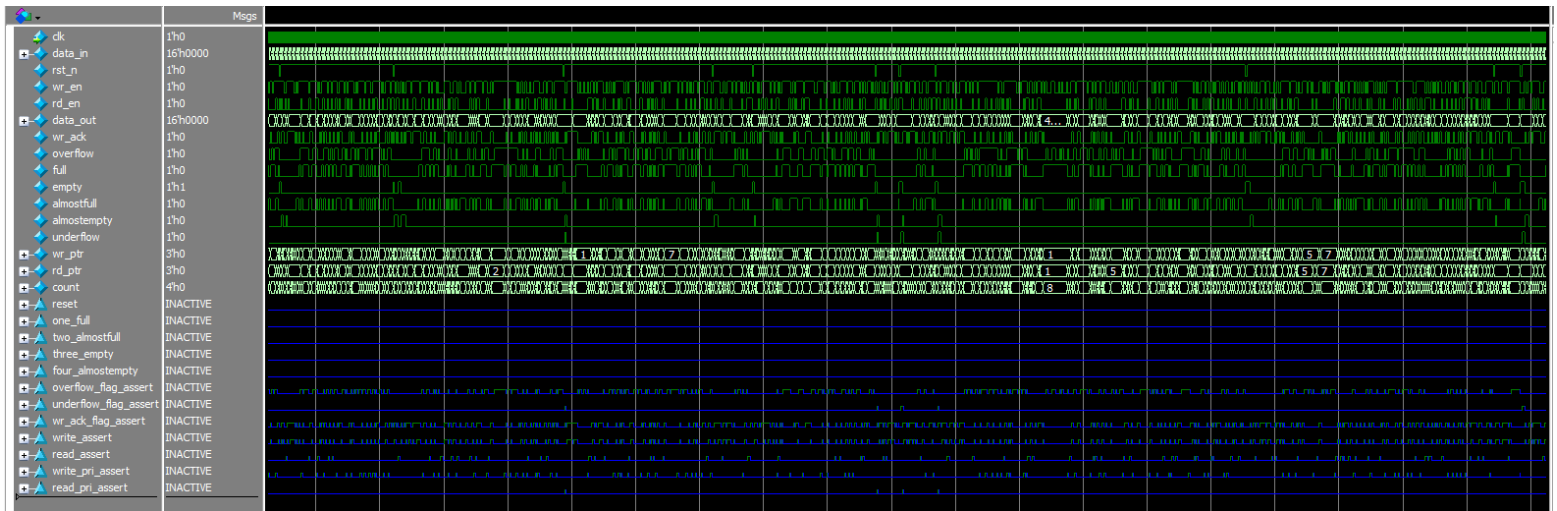
4. Coverage of assertions

Cover Directives										
Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included
+/FIFO_top/DUT/overflow_flag_cover	SVA	✓	Off	33007	1	Unli...	1	100%		✓
+/FIFO_top/DUT/underflow_flag_cover	SVA	✓	Off	509	1	Unli...	1	100%		✓
+/FIFO_top/DUT/wr_ack_flag_cover	SVA	✓	Off	35569	1	Unli...	1	100%		✓
+/FIFO_top/DUT/write_cover	SVA	✓	Off	24999	1	Unli...	1	100%		✓
+/FIFO_top/DUT/read_cover	SVA	✓	Off	8620	1	Unli...	1	100%		✓
+/FIFO_top/DUT/write_pri_cover	SVA	✓	Off	9911	1	Unli...	1	100%		✓
+/FIFO_top/DUT/read_pri_cover	SVA	✓	Off	354	1	Unli...	1	100%		✓

5. Covergroups

+/FIFO_coverage_pkg/FIFO_coverage	100.00%				
TYPE FIFO_cvgr	100.00%	100	100.00...		✓
CVP FIFO_cvgr::wr_en_cp	100.00%	100	100.00...		✓
CVP FIFO_cvgr::rd_en_cp	100.00%	100	100.00...		✓
CVP FIFO_cvgr::full_cp	100.00%	100	100.00...		✓
CVP FIFO_cvgr::empty_cp	100.00%	100	100.00...		✓
CVP FIFO_cvgr::almostfull_cp	100.00%	100	100.00...		✓
CVP FIFO_cvgr::almostempty_cp	100.00%	100	100.00...		✓
CVP FIFO_cvgr::overflow_cp	100.00%	100	100.00...		✓
CVP FIFO_cvgr::underflow_cp	100.00%	100	100.00...		✓
CVP FIFO_cvgr::wr_ack_cp	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::wr_full	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::wr_empty	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::wr_almostfull	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::wr_almostempty	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::wr_overflow	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::wr_underflow	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::wr_wr_ack	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::rd_full	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::rd_empty	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::rd_almostfull	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::rd_almostempty	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::rd_overflow	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::rd_underflow	100.00%	100	100.00...		✓
CROSS FIFO_cvgr::rd_wr_ack	100.00%	100	100.00...		✓

6. Full wave with assertions



VII. Coverage Report

1. Code Coverage (toggle, branch, statement)

```
=== Instance: /FIFO_top/f_if
=== Design Unit: work.FIFO_if
```

Toggle Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
Toggles	86	86	0	100.00%

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

Toggle Coverage for instance /FIFO_top/f_if --

Node	1H->0L	0L->1H	"Coverage"
almostempty	1	1	100.00
almostfull	1	1	100.00
clk	1	1	100.00
data_in[15-0]	1	1	100.00
data_out[15-0]	1	1	100.00
empty	1	1	100.00
full	1	1	100.00
overflow	1	1	100.00
rd_en	1	1	100.00
rst_n	1	1	100.00
underflow	1	1	100.00
wr_ack	1	1	100.00
wr_en	1	1	100.00

Branch Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
-----	----	----	-----	-----
Branches	35	35	0	100.00%

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

Branch Coverage for instance /FIFO_top/DUT

Statement Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
-----	----	----	-----	-----
Statements	29	29	0	100.00%

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

Statement Coverage for instance /FIFO_top/DUT --

2. Functional Coverage

COVERGROUP COVERAGE:

Covergroup	Metric	Goal	Bins	Status
-----	-----	-----	-----	-----
TYPE /FIFO_coverage_pkg/FIFO_coverage/FIFO_cvgr	100.00%	100	-	Covered
covered/total bins:	70	70	-	
missing/total bins:	0	70	-	
% Hit:	100.00%	100	-	

3. Assertions

=====				
=== Instance: /FIFO_top/DUT				
=== Design Unit: work.FIFO				
=====				
Assertion Coverage:				
Assertions	12	12	0	100.00%

Name	File(Line)	Failure Count	Pass Count	

/FIFO_top/DUT/reset	FIFO.sv(12)	0	1	
/FIFO_top/DUT/one_full	FIFO.sv(17)	0	1	
/FIFO_top/DUT/two_almostfull	FIFO.sv(20)	0	1	
/FIFO_top/DUT/three_empty	FIFO.sv(23)	0	1	
/FIFO_top/DUT/four_almostempty	FIFO.sv(26)	0	1	
/FIFO_top/DUT/overflow_flag_assert	FIFO.sv(32)	0	1	
/FIFO_top/DUT/underflow_flag_assert	FIFO.sv(38)	0	1	
/FIFO_top/DUT/wr_ack_flag_assert	FIFO.sv(44)	0	1	
/FIFO_top/DUT/write_assert	FIFO.sv(50)	0	1	
/FIFO_top/DUT/read_assert	FIFO.sv(56)	0	1	
/FIFO_top/DUT/write_pri_assert	FIFO.sv(62)	0	1	
/FIFO_top/DUT/read_pri_assert	FIFO.sv(68)	0	1	
Branch Coverage:				