Synchronous FIFO Verification

Created by:

Mohamed Ahmed Mohamed Hussein **26/9/24**

What is FIFO?

A FIFO (First-In, First-Out) is a type of data structure that behaves like a queue. The concept behind a FIFO is that the first data item entered is the first one to be removed, ensuring that the order of operations is preserved. This design is widely used in digital circuits, particularly in scenarios where data needs to be buffered or transferred between systems operating at different clock speeds.

Importance of FIFO in Digital Systems:

The primary importance of a FIFO lies in its ability to synchronize data between systems that do not share the same clock domain or have varying rates of data production and consumption. It acts as a buffer to hold the data temporarily until the receiving system is ready to process it. This makes FIFOs invaluable in designing communication interfaces, where data integrity and timing constraints are crucial.

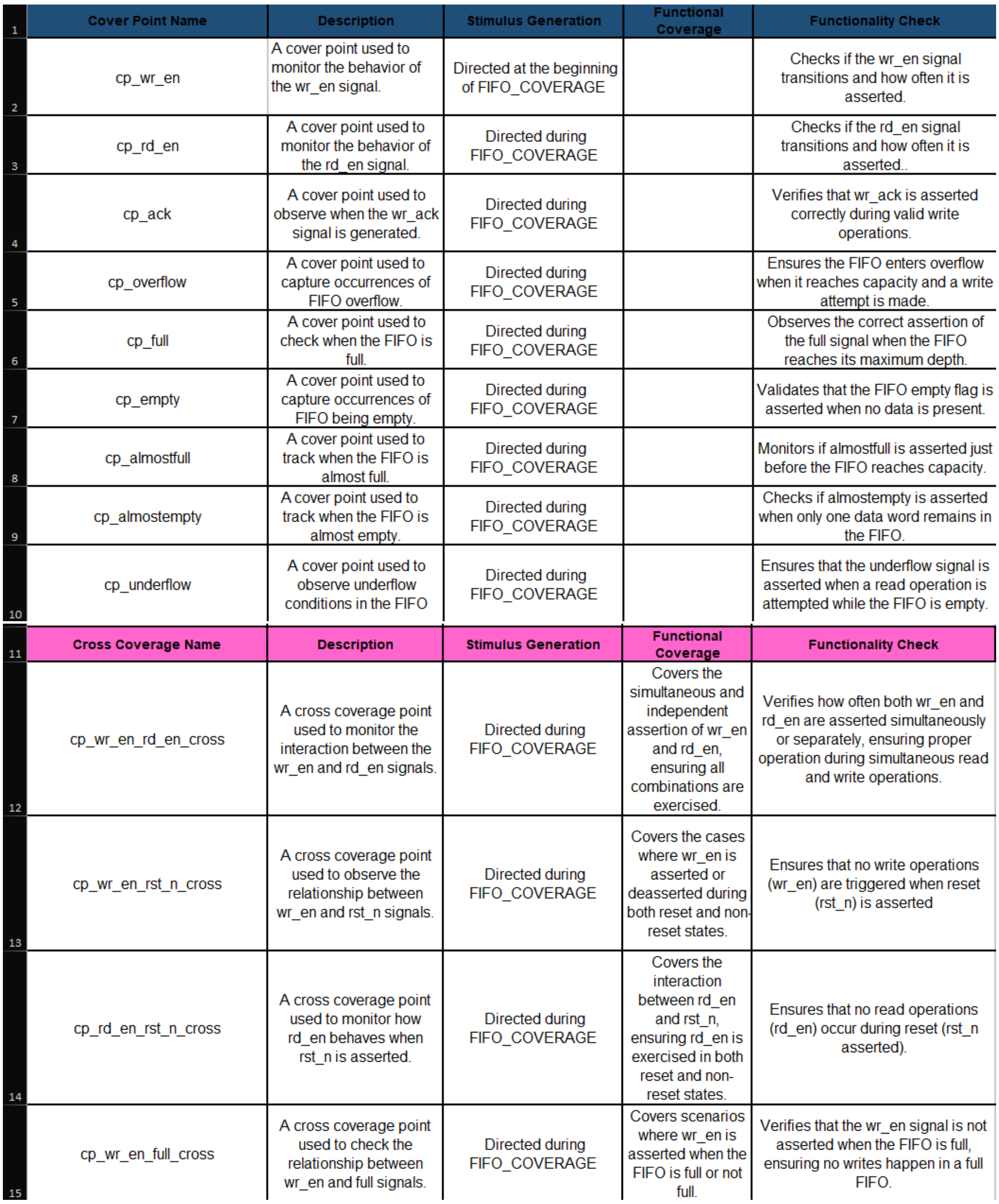
FIFOs also prevent data loss during high-speed data transfers by ensuring that there is always a place to store incoming data, even if the receiving system is momentarily busy. Furthermore, they support efficient data flow and reduce complexity in system design, improving overall performance.

Applications of FIFO:

1. **Clock Domain Crossing (CDC):** FIFOs are essential in systems where different clock domains exist, such as in multi-clock FPGAs or SoCs. They provide a way to transfer data across these domains safely without risking data corruption.
2. **Data Buffering in Communication Protocols:** FIFO buffers are used extensively in communication protocols like UART, SPI, and Ethernet, where they help smooth data transmission and reception across devices.
3. **Audio/Video Streaming:** In audio or video streaming, where real-time data flow is critical, FIFOs are used to handle continuous data streams, ensuring that data is processed sequentially and no frames are skipped.
4. **Processors and Microcontrollers:** Processors use FIFOs in their instruction pipelines and memory management units to handle instructions and data efficiently, allowing for smooth multitasking and efficient use of system resources.
5. **Networking:** In routers and switches, FIFOs help manage packet flows, ensuring that network packets are transmitted in the correct order and preventing packet loss in congested networks.
6. **Image Processing:** FIFOs are useful in image processing applications, where large sets of pixel data need to be buffered and processed sequentially to ensure accurate rendering.

FIFO Verification Report

Verification Plan:



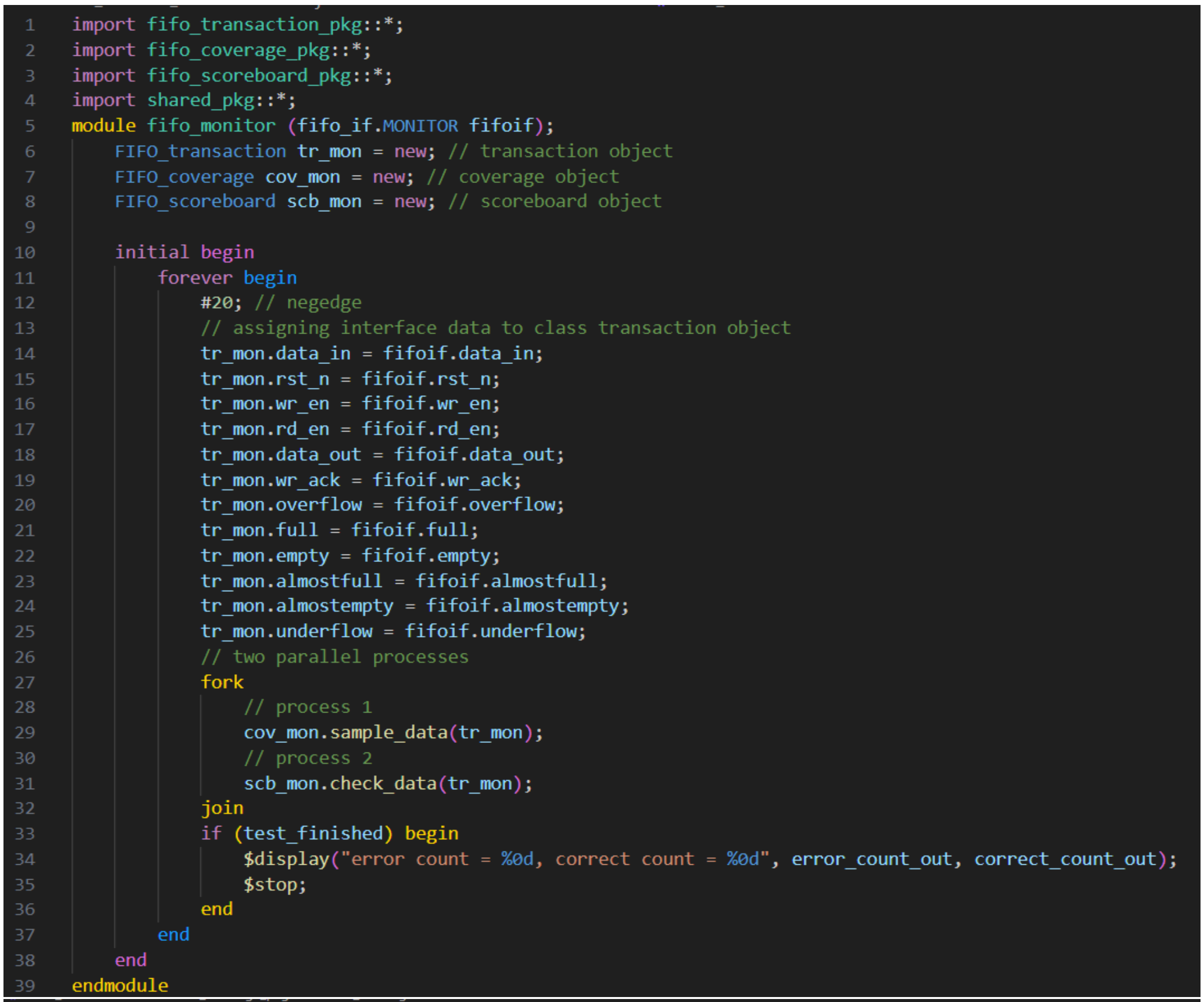
Cover Point Name	Description	Stimulus Generation	Functional Coverage	Functionality Check
cp_wr_en	A cover point used to monitor the behavior of the wr_en signal.	Directed at the beginning of FIFO_COVERAGE		Checks if the wr_en signal transitions and how often it is asserted.
cp_rd_en	A cover point used to monitor the behavior of the rd_en signal.	Directed during FIFO_COVERAGE		Checks if the rd_en signal transitions and how often it is asserted..
cp_ack	A cover point used to observe when the wr_ack signal is generated.	Directed during FIFO_COVERAGE		Verifies that wr_ack is asserted correctly during valid write operations.
cp_overflow	A cover point used to capture occurrences of FIFO overflow.	Directed during FIFO_COVERAGE		Ensures the FIFO enters overflow when it reaches capacity and a write attempt is made.
cp_full	A cover point used to check when the FIFO is full.	Directed during FIFO_COVERAGE		Observes the correct assertion of the full signal when the FIFO reaches its maximum depth.
cp_empty	A cover point used to capture occurrences of FIFO being empty.	Directed during FIFO_COVERAGE		Validates that the FIFO empty flag is asserted when no data is present.
cp_almostfull	A cover point used to track when the FIFO is almost full.	Directed during FIFO_COVERAGE		Monitors if almostfull is asserted just before the FIFO reaches capacity.
cp_almostempty	A cover point used to track when the FIFO is almost empty.	Directed during FIFO_COVERAGE		Checks if almostempty is asserted when only one data word remains in the FIFO.
cp_underflow	A cover point used to observe underflow conditions in the FIFO	Directed during FIFO_COVERAGE		Ensures that the underflow signal is asserted when a read operation is attempted while the FIFO is empty.
Cross Coverage Name	**Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
cp_wr_en_rd_en_cross	A cross coverage point used to monitor the interaction between the wr_en and rd_en signals.	Directed during FIFO_COVERAGE	Covers the simultaneous and independent assertion of wr_en and rd_en, ensuring all combinations are exercised.	Verifies how often both wr_en and rd_en are asserted simultaneously or separately, ensuring proper operation during simultaneous read and write operations.
cp_wr_en_rst_n_cross	A cross coverage point used to observe the relationship between wr_en and rst_n signals.	Directed during FIFO_COVERAGE	Covers the cases where wr_en is asserted or deasserted during both reset and non-reset states.	Ensures that no write operations (wr_en) are triggered when reset (rst_n) is asserted
cp_rd_en_rst_n_cross	A cross coverage point used to monitor how rd_en behaves when rst_n is asserted.	Directed during FIFO_COVERAGE	Covers the interaction between rd_en and rst_n, ensuring rd_en is exercised in both reset and non-reset states.	Ensures that no read operations (rd_en) occur during reset (rst_n asserted).
cp_wr_en_full_cross	A cross coverage point used to check the relationship between wr_en and full signals.	Directed during FIFO_COVERAGE	Covers scenarios where wr_en is asserted when the FIFO is full or not full.	Verifies that the wr_en signal is not asserted when the FIFO is full, ensuring no writes happen in a full FIFO.

16	cp_rd_en_empty_cross	A cross coverage point used to track the interaction between rd_en and empty signals.	Directed during FIFO_COVERAGE	Covers the behavior of rd_en when the FIFO is empty or not empty.	Ensures that no read operations (rd_en) occur when the FIFO is empty, preventing underflow scenarios.
17	cp_wr_ack_wr_en_cross	A cross coverage point used to monitor the relationship between wr_ack and wr_en signals.	Directed during FIFO_COVERAGE	Covers the different combinations of wr_en and wr_ack, ensuring correct handshaking between write operations.	Verifies that wr_ack is correctly asserted when valid write operations (wr_en) are performed.
18	cp_rd_en_underflow_cross	A cross coverage point used to track how rd_en behaves when underflow conditions occur.	Directed during FIFO_COVERAGE	Covers scenarios where rd_en is asserted in both underflow and normal conditions.	Ensures that underflow situations are properly handled when rd_en is asserted while the FIFO is empty.
19	cp_full_almostfull_cross	A cross coverage point used to observe the relationship between full and almostfull signals.	Directed during FIFO_COVERAGE	Covers the transition from almostfull to full, ensuring all boundary conditions are checked.	Verifies that the almostfull signal is asserted just before full is reached, ensuring correct behavior near full capacity.
20	cp_empty_almostempty_cross	A cross coverage point used to monitor the transition between empty and almostempty signals.	Directed during FIFO_COVERAGE	Covers the transition between almostempty and empty, ensuring proper indication as the FIFO empties.	Ensures that almostempty is asserted when only one word remains, and empty is asserted when no data is left
21	cp_overflow_wr_en_cross	A cross coverage point used to capture overflow conditions when wr_en is asserted during full conditions.	Directed at the end of FIFO_COVERAGE	Covers the scenario where wr_en is asserted when the FIFO is full, checking for correct overflow detection.	Verifies that overflow only happens when wr_en is asserted while the FIFO is full, ensuring proper handling of this scenario.

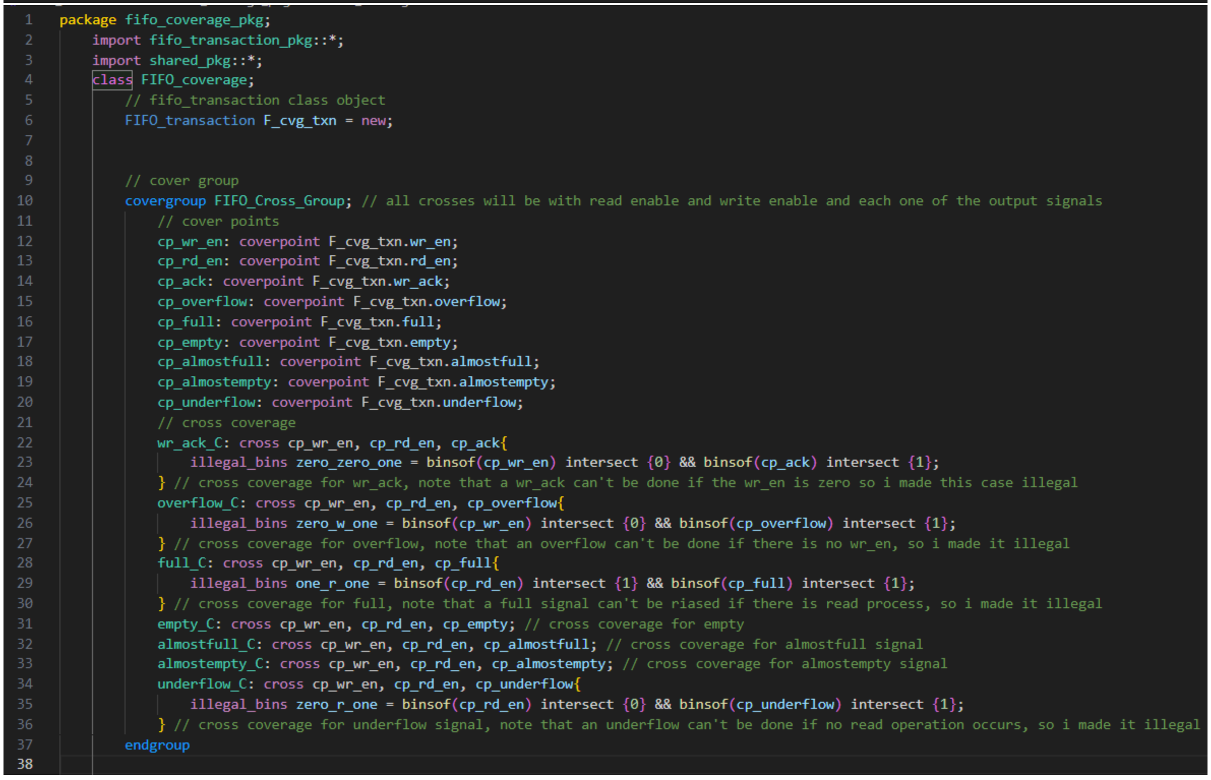
22	Assertion Name	Description	Stimulus Generation	Functional Coverage	Functionality Check
23	reset assertion	An immediate assertion used to verify that signals are correctly reset when rst_n is deasserted.	Directed at the beginning of the design code (FIFO)		Ensures that all internal signals such as count, wr_ptr, rd_ptr, full, empty, etc., are correctly reset.
24	full_a	An assertion used to check that the full signal is asserted when the FIFO depth is reached.	Directed during the design code (FIFO)		Validates that the FIFO full flag is asserted when the FIFO reaches its maximum capacity.
25	empty_a	An assertion used to verify that the FIFO's empty signal is asserted when no data is present.	Directed during the design code (FIFO)		Confirms that the FIFO is indeed empty when the empty flag is high.
26	almostfull_a	An assertion used to check that the almostfull signal is correctly asserted when the FIFO is nearly full.	Directed during the design code (FIFO)		Verifies that the FIFO is nearing capacity when almostfull is asserted.
27	almostempty_a	An assertion used to check that the almostempty signal is asserted when only one data word is left in the FIFO.	Directed during the design code (FIFO)		Ensures that the FIFO is nearly empty when almostempty is correctly asserted.

28	ack_a	An assertion used to ensure that wr_ack is generated only when a write operation is successful.	Directed during the design code (FIFO)		Verifies that wr_ack is asserted during valid write operations.
29	overflow_a	An assertion used to detect when the FIFO overflows upon attempting a write operation while full.	Directed during the design code (FIFO)		Ensures that the FIFO enters overflow only when the FIFO is full and a write attempt is made.
30	underflow_a	An assertion used to catch underflow situations when attempting a read operation on an empty FIFO.	Directed during the design code (FIFO)		Validates that the FIFO enters underflow only when a read operation is attempted while the FIFO is empty.
31	wr_ptr_a	An assertion used to track the wr_ptr increment after valid write operations.	Directed during the design code (FIFO)		Ensures that the write pointer increments correctly with every valid write operation.
32	rd_ptr_a	An assertion used to track the rd_ptr increment after valid read operations.	Directed during the design code (FIFO)		Ensures that the read pointer increments correctly with every valid read operation.
33	count_write_priority_a	An assertion used to verify the counter behavior when write and read enable signals are both active with empty.	Directed during the design code (FIFO)		Confirms that the counter increments as expected in this priority case.
34	count_read_priority_a	An assertion used to verify the counter behavior when both write and read are enabled with the FIFO full.	Directed during the design code (FIFO)		Confirms that the counter decrements as expected in this priority case.
35	count_w_a	An assertion used to verify that the counter increments with valid writes and no read operations.	Directed during the design code (FIFO)		Ensures that the counter increments correctly when a valid write occurs and no read is happening.
36	count_r_a	An assertion used to verify that the counter decrements with valid reads and no write operations.	Directed during the design code (FIFO)		Ensures that the counter decrements correctly when a valid read occurs and no write is happening.
37	**Constraint Name**	**Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
38	reset_con	A constraint used to control the distribution of rst_n, making reset occur less frequently.	Directed at the end of FIFO_TRANSACTION		Ensures that rst_n is mostly high with rare assertions, simulating rare reset conditions during test runs.
39	wr_en_con	A constraint used to define the probability of wr_en being asserted based on WR_EN_ON_DIST.	Directed at the end of FIFO_TRANSACTION		Verifies that the wr_en signal follows the expected distribution, ensuring write operations occur at the correct frequency
40	rd_en_con	A constraint used to define the probability of rd_en being asserted based on RD_EN_ON_DIST.	Directed at the end of FIFO_TRANSACTION		Verifies that the rd_en signal follows the expected distribution, ensuring read operations occur at the correct frequency.

Snippet of Monitor and Coverage Codes:



```
import fifo_transaction_pkg::*;
import fifo_coverage_pkg::*;
import fifo_scoreboard_pkg::*;
import shared_pkg::*;
module fifo_monitor (fifo_if.MONITOR fifoif);
    FIFO_transaction tr_mon = new; // transaction object
    FIFO_coverage cov_mon = new; // coverage object
    FIFO_scoreboard scb_mon = new; // scoreboard object

    initial begin
        forever begin
            #20; // negedge
            // assigning interface data to class transaction object
            tr_mon.data_in = fifoif.data_in;
            tr_mon.rst_n = fifoif.rst_n;
            tr_mon.wr_en = fifoif.wr_en;
            tr_mon.rd_en = fifoif.rd_en;
            tr_mon.data_out = fifoif.data_out;
            tr_mon.wr_ack = fifoif.wr_ack;
            tr_mon.overflow = fifoif.overflow;
            tr_mon.full = fifoif.full;
            tr_mon.empty = fifoif.empty;
            tr_mon.almostfull = fifoif.almostfull;
            tr_mon.almostempty = fifoif.almostempty;
            tr_mon.underflow = fifoif.underflow;
            // two parallel processes
            fork
                // process 1
                cov_mon.sample_data(tr_mon);
                // process 2
                scb_mon.check_data(tr_mon);
            join
            if (test_finished) begin
                $display("error count = %0d, correct count = %0d", error_count_out, correct_count_out);
                $stop;
            end
        end
    end
endmodule
```



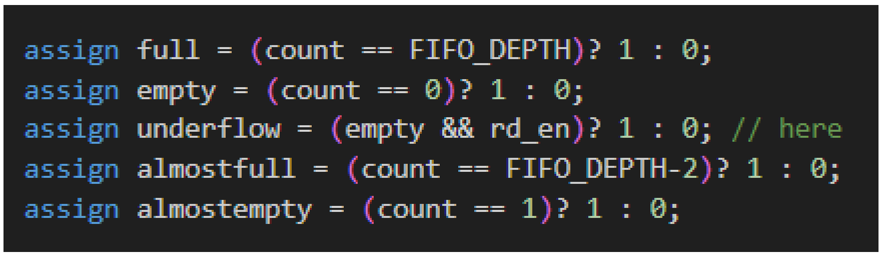
```
package fifo_coverage_pkg;
    import fifo_transaction_pkg::*;
    import shared_pkg::*;
    class FIFO_coverage;
        // fifo_transaction class object
        FIFO_transaction F_cvg_txn = new;


        // cover group
        covergroup FIFO_Cross_Group; // all crosses will be with read enable and write enable and each one of the output signals
            // cover points
            cp_wr_en: coverpoint F_cvg_txn.wr_en;
            cp_rd_en: coverpoint F_cvg_txn.rd_en;
            cp_ack: coverpoint F_cvg_txn.wr_ack;
            cp_overflow: coverpoint F_cvg_txn.overflow;
            cp_full: coverpoint F_cvg_txn.full;
            cp_empty: coverpoint F_cvg_txn.empty;
            cp_almostfull: coverpoint F_cvg_txn.almostfull;
            cp_almostempty: coverpoint F_cvg_txn.almostempty;
            cp_underflow: coverpoint F_cvg_txn.underflow;
            // cross coverage
            wr_ack_C: cross cp_wr_en, cp_rd_en, cp_ack{
                illegal_bins zero_zero_one = binsof(cp_wr_en) intersect {0} && binsof(cp_ack) intersect {1};
            } // cross coverage for wr_ack, note that a wr_ack can't be done if the wr_en is zero so i made this case illegal
            overflow_C: cross cp_wr_en, cp_rd_en, cp_overflow{
                illegal_bins zero_w_one = binsof(cp_wr_en) intersect {0} && binsof(cp_overflow) intersect {1};
            } // cross coverage for overflow, note that an overflow can't be done if there is no wr_en, so i made it illegal
            full_C: cross cp_wr_en, cp_rd_en, cp_full{
                illegal_bins one_r_one = binsof(cp_rd_en) intersect {1} && binsof(cp_full) intersect {1};
            } // cross coverage for full, note that a full signal can't be riased if there is read process, so i made it illegal
            empty_C: cross cp_wr_en, cp_rd_en, cp_empty; // cross coverage for empty
            almostfull_C: cross cp_wr_en, cp_rd_en, cp_almostfull; // cross coverage for almostfull signal
            almostempty_C: cross cp_wr_en, cp_rd_en, cp_almostempty; // cross coverage for almostempty signal
            underflow_C: cross cp_wr_en, cp_rd_en, cp_underflow{
                illegal_bins zero_r_one = binsof(cp_rd_en) intersect {0} && binsof(cp_underflow) intersect {1};
            } // cross coverage for underflow signal, note that an underflow can't be done if no read operation occurs, so i made it illegal
        endgroup

```

Bugs Report:

- **Underflow signal bug:**

  Bug: underflow signal was meant to be sequential (following clock edge) in the Specs

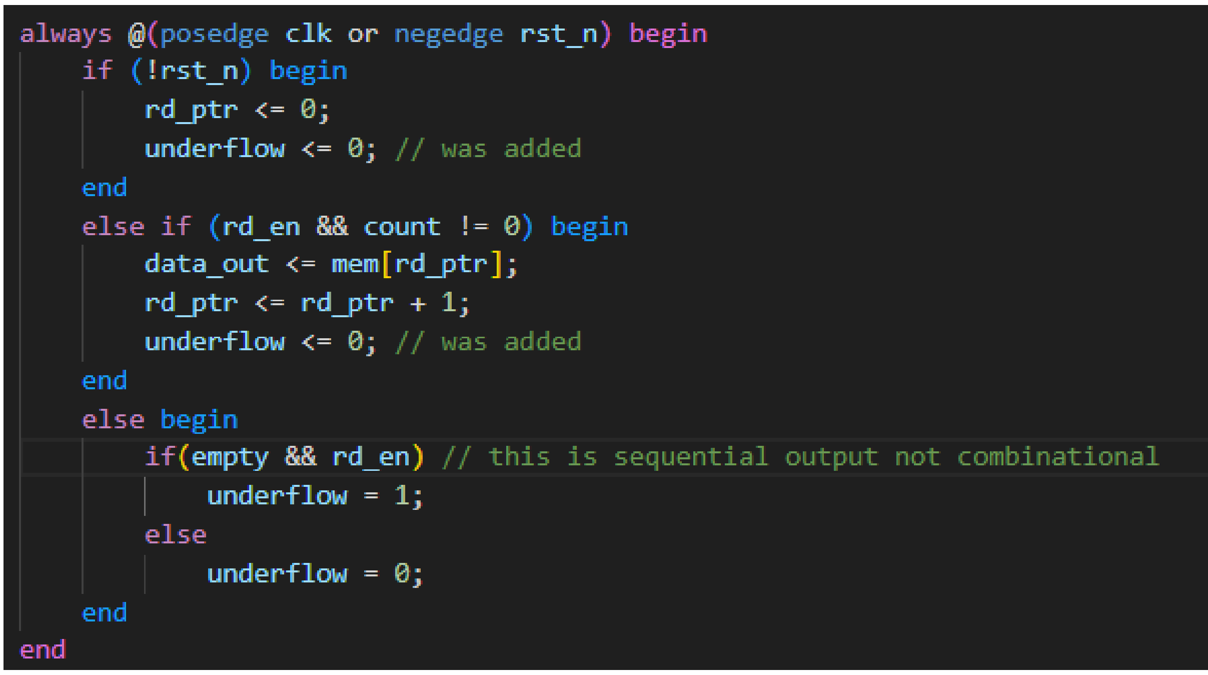
  Original design:



```
assign full = (count == FIFO_DEPTH)? 1 : 0;
assign empty = (count == 0)? 1 : 0;
assign underflow = (empty && rd_en)? 1 : 0; // here
assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
assign almostempty = (count == 1)? 1 : 0;
```

  Modification: I made it following clock edge in the read always block

  Modified design:

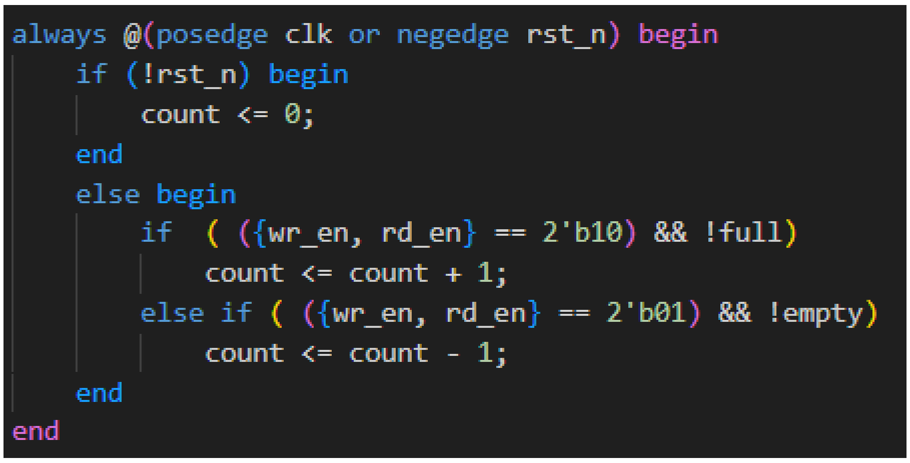


```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rd_ptr <= 0;
        underflow <= 0; // was added
    end
    else if (rd_en && count != 0) begin
        data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        underflow <= 0; // was added
    end
    else begin
        if(empty && rd_en) // this is sequential output not combinational
            underflow = 1;
        else
            underflow = 0;
    end
end
```

- **Missing conditions:**

  Bug: the Spec was “If a read and write enables were high and the FIFO was empty, only writing will take place and vice verse if the FIFO was full” and it was not implemented in the original design

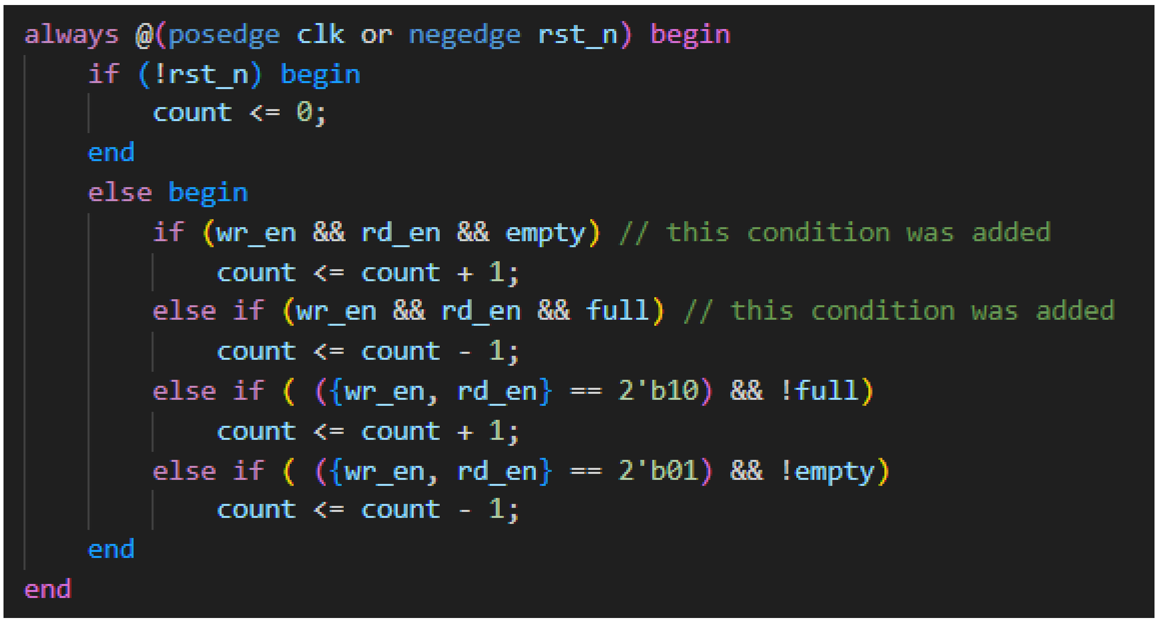
  Original design:



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

  Modification: I added the right conditions to meet the Specs

  Modified design:

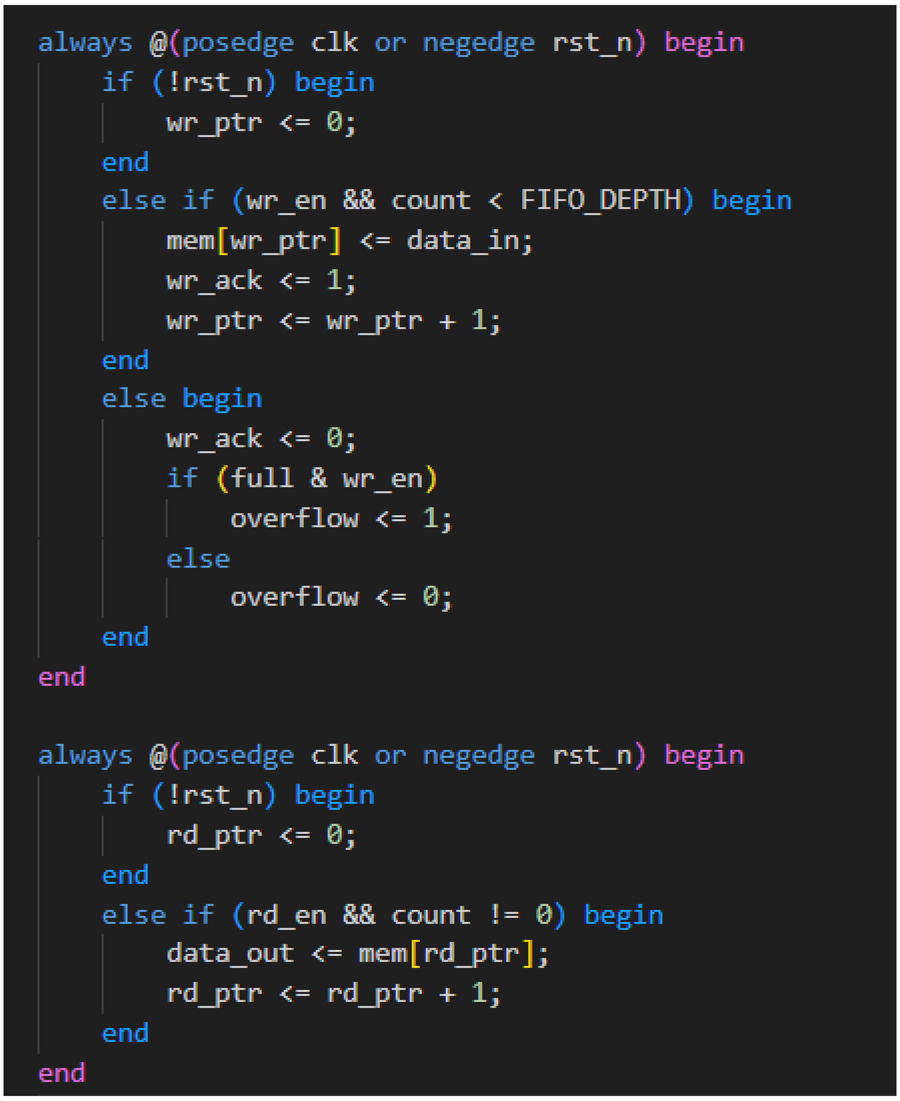


```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        count <= 0;
    end
    else begin
        if (wr_en && rd_en && empty) // this condition was added
            count <= count + 1;
        else if (wr_en && rd_en && full) // this condition was added
            count <= count - 1;
        else if ( ({wr_en, rd_en} == 2'b10) && !full)
            count <= count + 1;
        else if ( ({wr_en, rd_en} == 2'b01) && !empty)
            count <= count - 1;
    end
end
```

- **Unknown behavior while reset:**

  Bug: there are some signals which their output haven't been specified when reset signal takes place

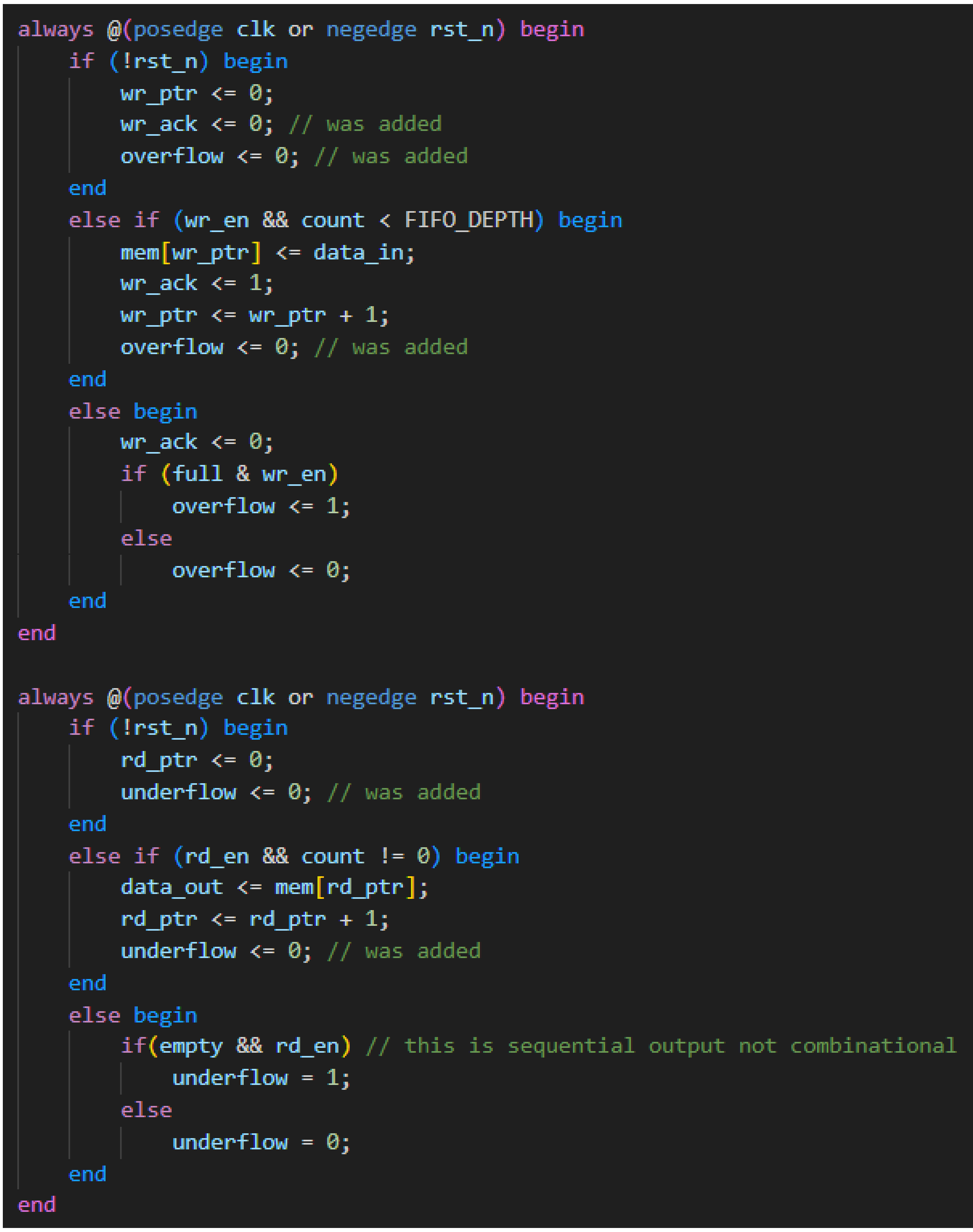
  Original design:



```verilog
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        wr_ptr <= 0;
    end
    else if (wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= data_in;
        wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    end
    else begin
        wr_ack <= 0;
        if (full & wr_en)
            overflow <= 1;
        else
            overflow <= 0;
    end
end

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

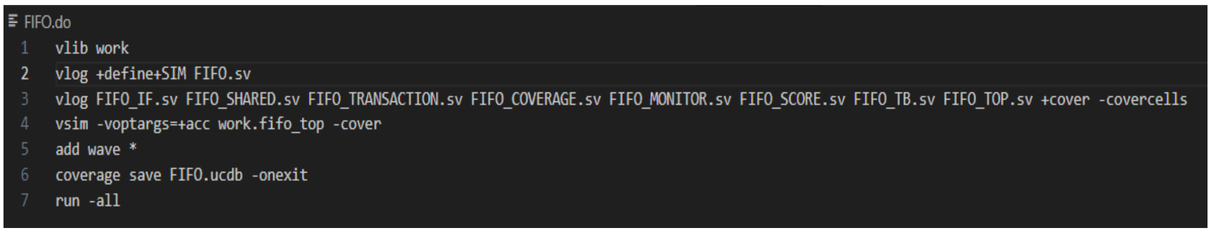
  Modification: I added the right conditions to meet the Specs

Modified design:



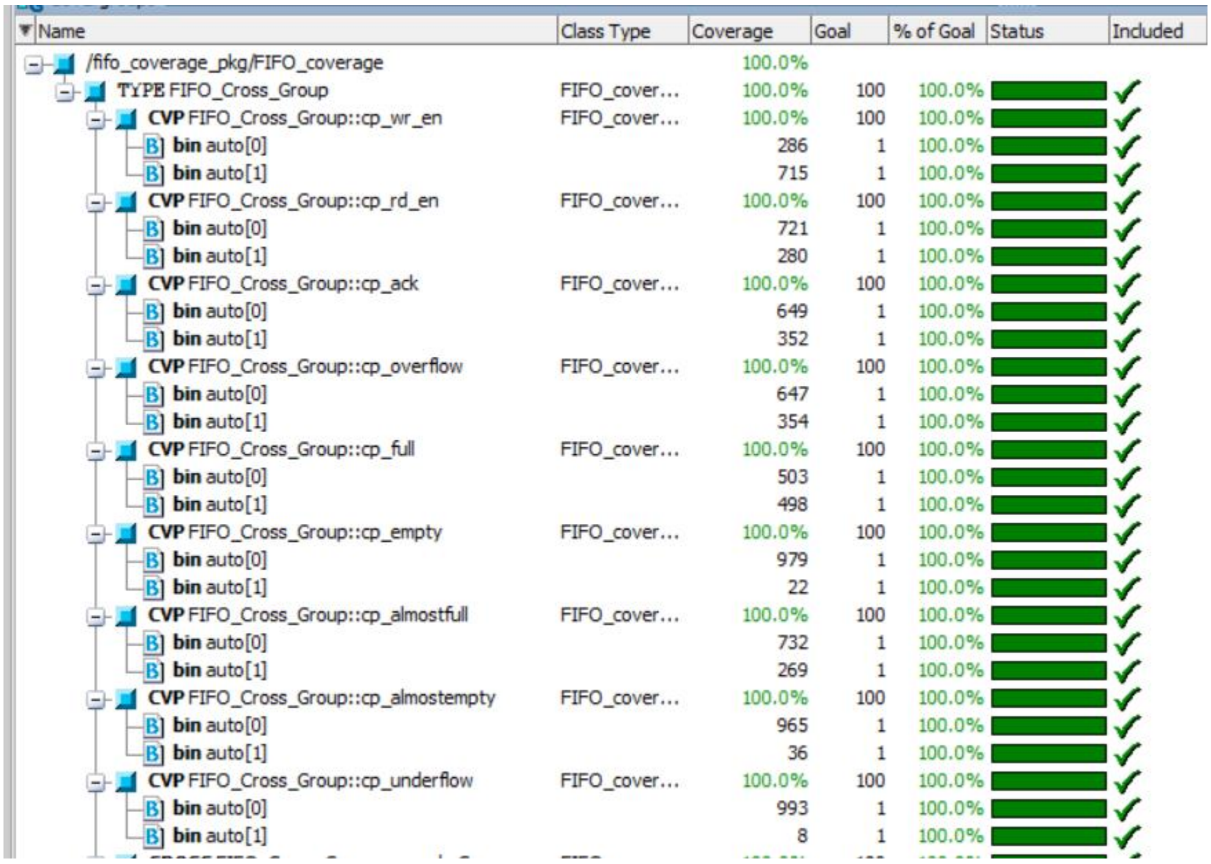
```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        wr_ptr <= 0;
        wr_ack <= 0; // was added
        overflow <= 0; // was added
    end
    else if (wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= data_in;
        wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
        overflow <= 0; // was added
    end
    else begin
        wr_ack <= 0;
        if (full & wr_en)
            overflow <= 1;
        else
            overflow <= 0;
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rd_ptr <= 0;
        underflow <= 0; // was added
    end
    else if (rd_en && count != 0) begin
        data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        underflow <= 0; // was added
    end
    else begin
        if(empty && rd_en) // this is sequential output not combinational
            underflow = 1;
        else
            underflow = 0;
    end
end
```

Do File:

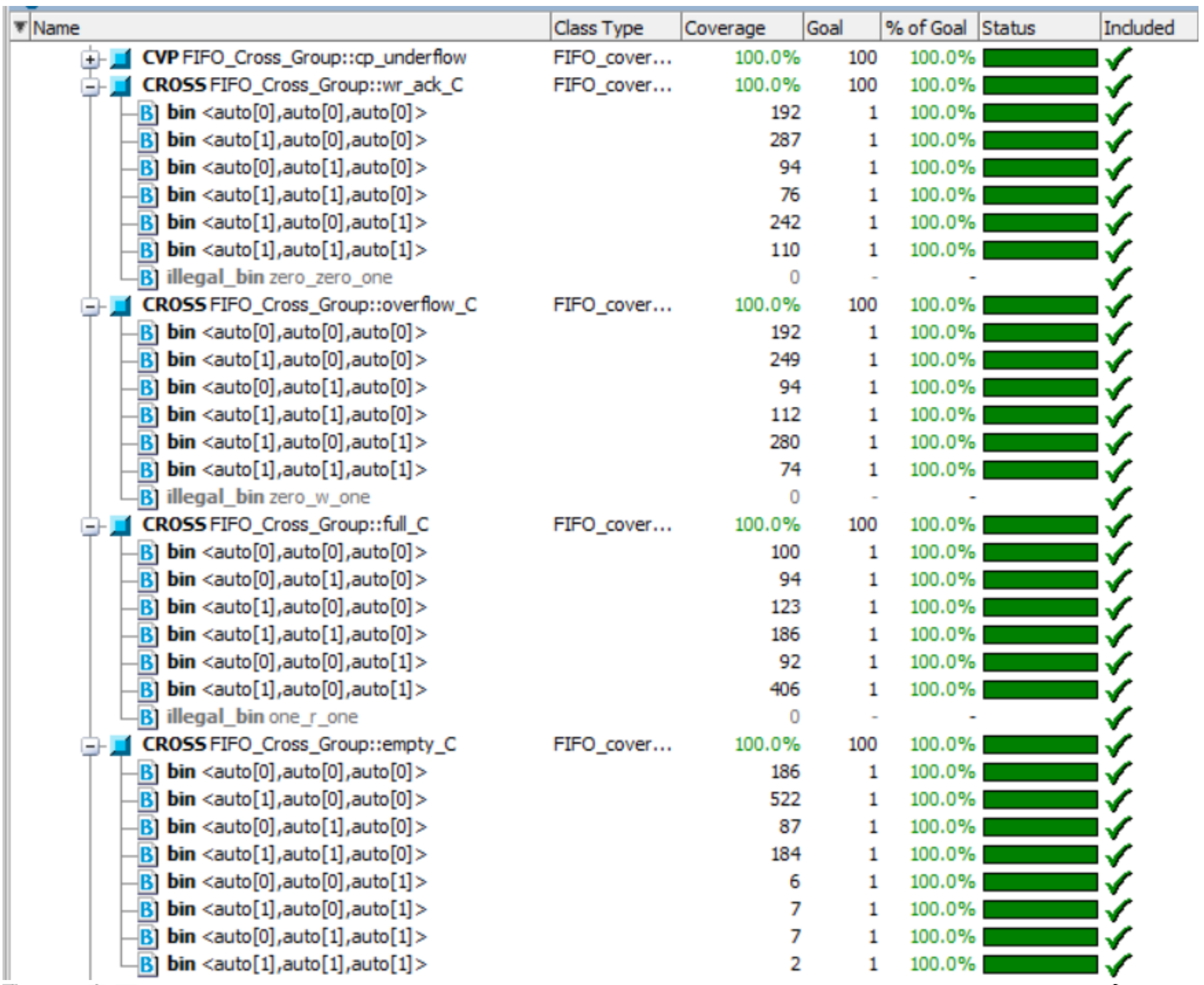


```
FIFO.do
vlib work
vlog +define+SIM FIFO.sv
vlog FIFO_IF.sv FIFO_SHARED.sv FIFO_TRANSACTION.sv FIFO_COVERAGE.sv FIFO_MONITOR.sv FIFO_SCORE.sv FIFO_TB.sv FIFO_TOP.sv +cover -covercells
vsim -voptargs=+acc work.fifo_top -cover
add wave *
coverage save FIFO.ucdb -onexit
run -all
```

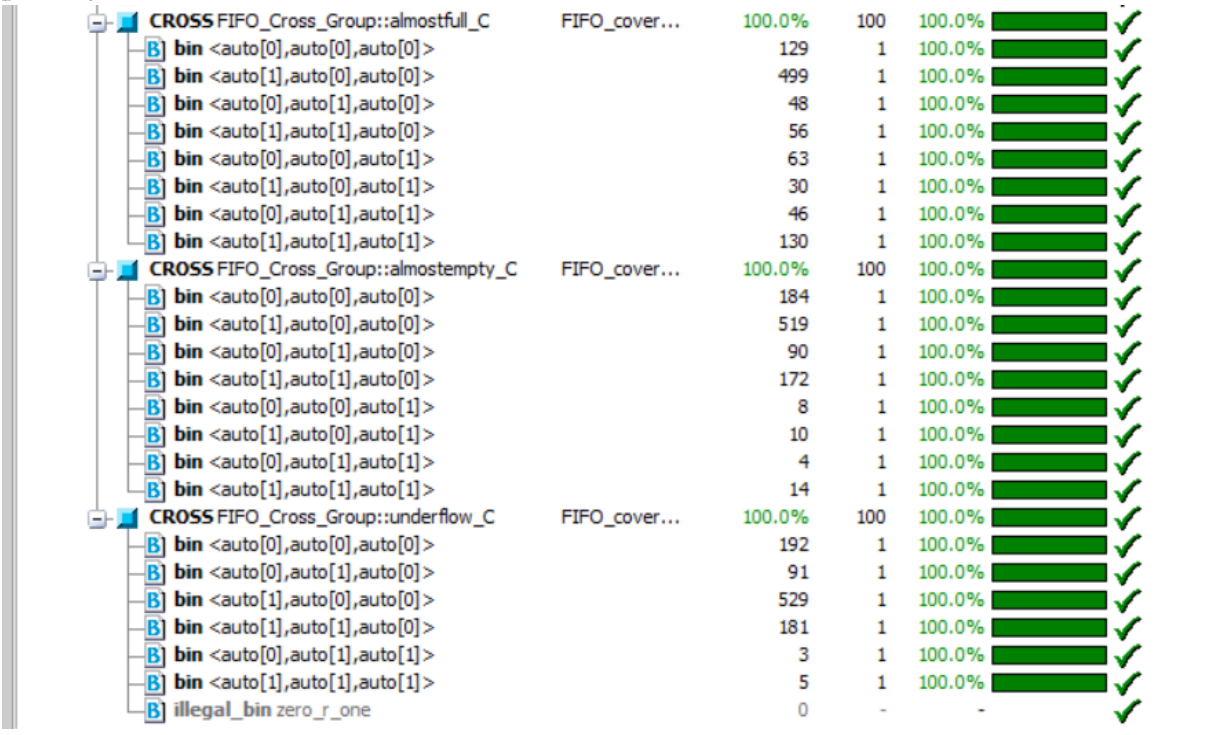
Functional Coverage (including cross coverage):



Name	Class Type	Coverage	Goal	% of Goal	Status	Included
/fifo_coverage_pkg/FIFO_coverage		100.0%				
TYPE FIFO_Cross_Group	FIFO_cover...	100.0%	100	100.0%	🟩	✓
CVP FIFO_Cross_Group::cp_wr_en	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin auto[0]		286	1	100.0%	🟩	✓
bin auto[1]		715	1	100.0%	🟩	✓
CVP FIFO_Cross_Group::cp_rd_en	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin auto[0]		721	1	100.0%	🟩	✓
bin auto[1]		280	1	100.0%	🟩	✓
CVP FIFO_Cross_Group::cp_ack	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin auto[0]		649	1	100.0%	🟩	✓
bin auto[1]		352	1	100.0%	🟩	✓
CVP FIFO_Cross_Group::cp_overflow	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin auto[0]		647	1	100.0%	🟩	✓
bin auto[1]		354	1	100.0%	🟩	✓
CVP FIFO_Cross_Group::cp_full	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin auto[0]		503	1	100.0%	🟩	✓
bin auto[1]		498	1	100.0%	🟩	✓
CVP FIFO_Cross_Group::cp_empty	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin auto[0]		979	1	100.0%	🟩	✓
bin auto[1]		22	1	100.0%	🟩	✓
CVP FIFO_Cross_Group::cp_almostfull	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin auto[0]		732	1	100.0%	🟩	✓
bin auto[1]		269	1	100.0%	🟩	✓
CVP FIFO_Cross_Group::cp_almostempty	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin auto[0]		965	1	100.0%	🟩	✓
bin auto[1]		36	1	100.0%	🟩	✓
CVP FIFO_Cross_Group::cp_underflow	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin auto[0]		993	1	100.0%	🟩	✓
bin auto[1]		8	1	100.0%	🟩	✓

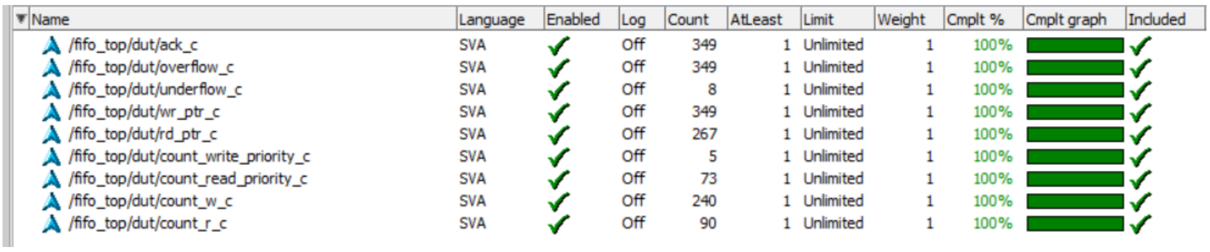


Name	Class Type	Coverage	Goal	% of Goal	Status	Included
CVP FIFO_Cross_Group::cp_underflow	FIFO_cover...	100.0%	100	100.0%	🟩	✓
CROSS FIFO_Cross_Group::wr_ack_C	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin <auto[0],auto[0],auto[0]>		192	1	100.0%	🟩	✓
bin <auto[1],auto[0],auto[0]>		287	1	100.0%	🟩	✓
bin <auto[0],auto[1],auto[0]>		94	1	100.0%	🟩	✓
bin <auto[1],auto[1],auto[0]>		76	1	100.0%	🟩	✓
bin <auto[1],auto[0],auto[1]>		242	1	100.0%	🟩	✓
bin <auto[1],auto[1],auto[1]>		110	1	100.0%	🟩	✓
illegal_bin zero_zero_one		0	-	-		✓
CROSS FIFO_Cross_Group::overflow_C	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin <auto[0],auto[0],auto[0]>		192	1	100.0%	🟩	✓
bin <auto[1],auto[0],auto[0]>		249	1	100.0%	🟩	✓
bin <auto[0],auto[1],auto[0]>		94	1	100.0%	🟩	✓
bin <auto[1],auto[1],auto[0]>		112	1	100.0%	🟩	✓
bin <auto[1],auto[0],auto[1]>		280	1	100.0%	🟩	✓
bin <auto[1],auto[1],auto[1]>		74	1	100.0%	🟩	✓
illegal_bin zero_w_one		0	-	-		✓
CROSS FIFO_Cross_Group::full_C	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin <auto[0],auto[0],auto[0]>		100	1	100.0%	🟩	✓
bin <auto[0],auto[1],auto[0]>		94	1	100.0%	🟩	✓
bin <auto[1],auto[0],auto[0]>		123	1	100.0%	🟩	✓
bin <auto[1],auto[1],auto[0]>		186	1	100.0%	🟩	✓
bin <auto[0],auto[0],auto[1]>		92	1	100.0%	🟩	✓
bin <auto[1],auto[0],auto[1]>		406	1	100.0%	🟩	✓
illegal_bin one_r_one		0	-	-		✓
CROSS FIFO_Cross_Group::empty_C	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin <auto[0],auto[0],auto[0]>		186	1	100.0%	🟩	✓
bin <auto[1],auto[0],auto[0]>		522	1	100.0%	🟩	✓
bin <auto[0],auto[1],auto[0]>		87	1	100.0%	🟩	✓
bin <auto[1],auto[1],auto[0]>		184	1	100.0%	🟩	✓
bin <auto[0],auto[0],auto[1]>		6	1	100.0%	🟩	✓
bin <auto[1],auto[0],auto[1]>		7	1	100.0%	🟩	✓
bin <auto[0],auto[1],auto[1]>		7	1	100.0%	🟩	✓
bin <auto[1],auto[1],auto[1]>		2	1	100.0%	🟩	✓



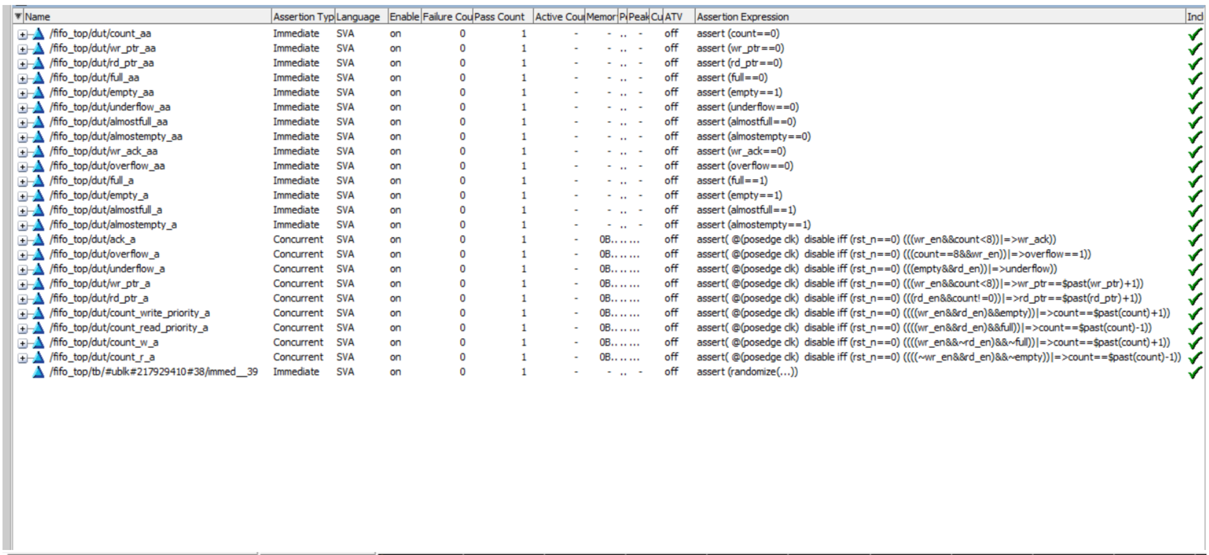
CROSS FIFO_Cross_Group::almostfull_C	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin <auto[0],auto[0],auto[0]>		129	1	100.0%	🟩	✓
bin <auto[1],auto[0],auto[0]>		499	1	100.0%	🟩	✓
bin <auto[0],auto[1],auto[0]>		48	1	100.0%	🟩	✓
bin <auto[1],auto[1],auto[0]>		56	1	100.0%	🟩	✓
bin <auto[0],auto[0],auto[1]>		63	1	100.0%	🟩	✓
bin <auto[1],auto[0],auto[1]>		30	1	100.0%	🟩	✓
bin <auto[0],auto[1],auto[1]>		46	1	100.0%	🟩	✓
bin <auto[1],auto[1],auto[1]>		130	1	100.0%	🟩	✓
CROSS FIFO_Cross_Group::almostempty_C	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin <auto[0],auto[0],auto[0]>		184	1	100.0%	🟩	✓
bin <auto[1],auto[0],auto[0]>		519	1	100.0%	🟩	✓
bin <auto[0],auto[1],auto[0]>		90	1	100.0%	🟩	✓
bin <auto[1],auto[1],auto[0]>		172	1	100.0%	🟩	✓
bin <auto[0],auto[0],auto[1]>		8	1	100.0%	🟩	✓
bin <auto[1],auto[0],auto[1]>		10	1	100.0%	🟩	✓
bin <auto[0],auto[1],auto[1]>		4	1	100.0%	🟩	✓
bin <auto[1],auto[1],auto[1]>		14	1	100.0%	🟩	✓
CROSS FIFO_Cross_Group::underflow_C	FIFO_cover...	100.0%	100	100.0%	🟩	✓
bin <auto[0],auto[0],auto[0]>		192	1	100.0%	🟩	✓
bin <auto[0],auto[1],auto[0]>		91	1	100.0%	🟩	✓
bin <auto[1],auto[0],auto[0]>		529	1	100.0%	🟩	✓
bin <auto[1],auto[1],auto[0]>		181	1	100.0%	🟩	✓
bin <auto[0],auto[1],auto[1]>		3	1	100.0%	🟩	✓
bin <auto[1],auto[1],auto[1]>		5	1	100.0%	🟩	✓
illegal_bin zero_r_one		0	-	-		✓

Assertions Coverage:

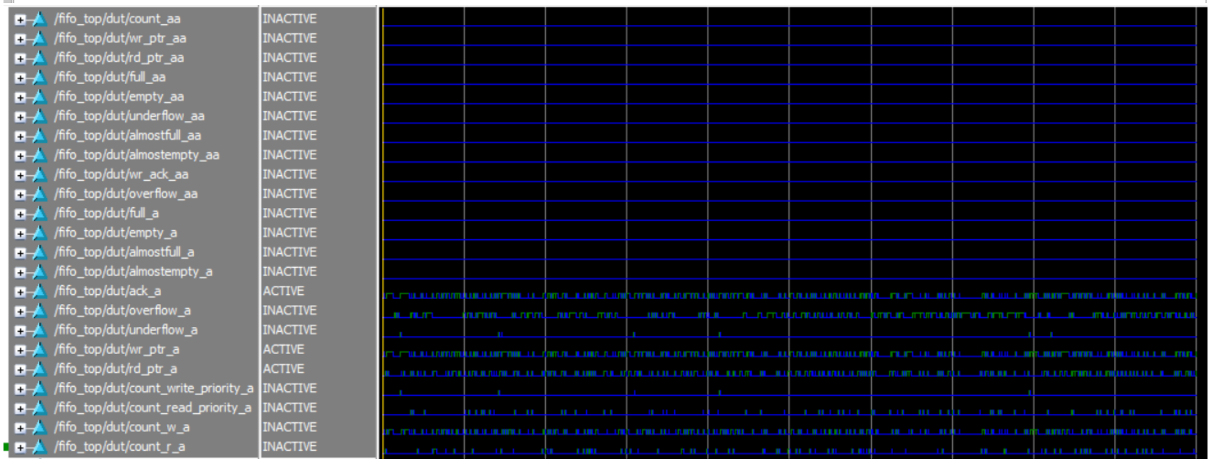


Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included
/fifo_top/dut/ack_c	SVA	✓	Off	349	1	Unlimited	1	100%	🟩	✓
/fifo_top/dut/overflow_c	SVA	✓	Off	349	1	Unlimited	1	100%	🟩	✓
/fifo_top/dut/underflow_c	SVA	✓	Off	8	1	Unlimited	1	100%	🟩	✓
/fifo_top/dut/wr_ptr_c	SVA	✓	Off	349	1	Unlimited	1	100%	🟩	✓
/fifo_top/dut/rd_ptr_c	SVA	✓	Off	267	1	Unlimited	1	100%	🟩	✓
/fifo_top/dut/count_write_priority_c	SVA	✓	Off	5	1	Unlimited	1	100%	🟩	✓
/fifo_top/dut/count_read_priority_c	SVA	✓	Off	73	1	Unlimited	1	100%	🟩	✓
/fifo_top/dut/count_w_c	SVA	✓	Off	240	1	Unlimited	1	100%	🟩	✓
/fifo_top/dut/count_r_c	SVA	✓	Off	90	1	Unlimited	1	100%	🟩	✓

Assertions Passed:



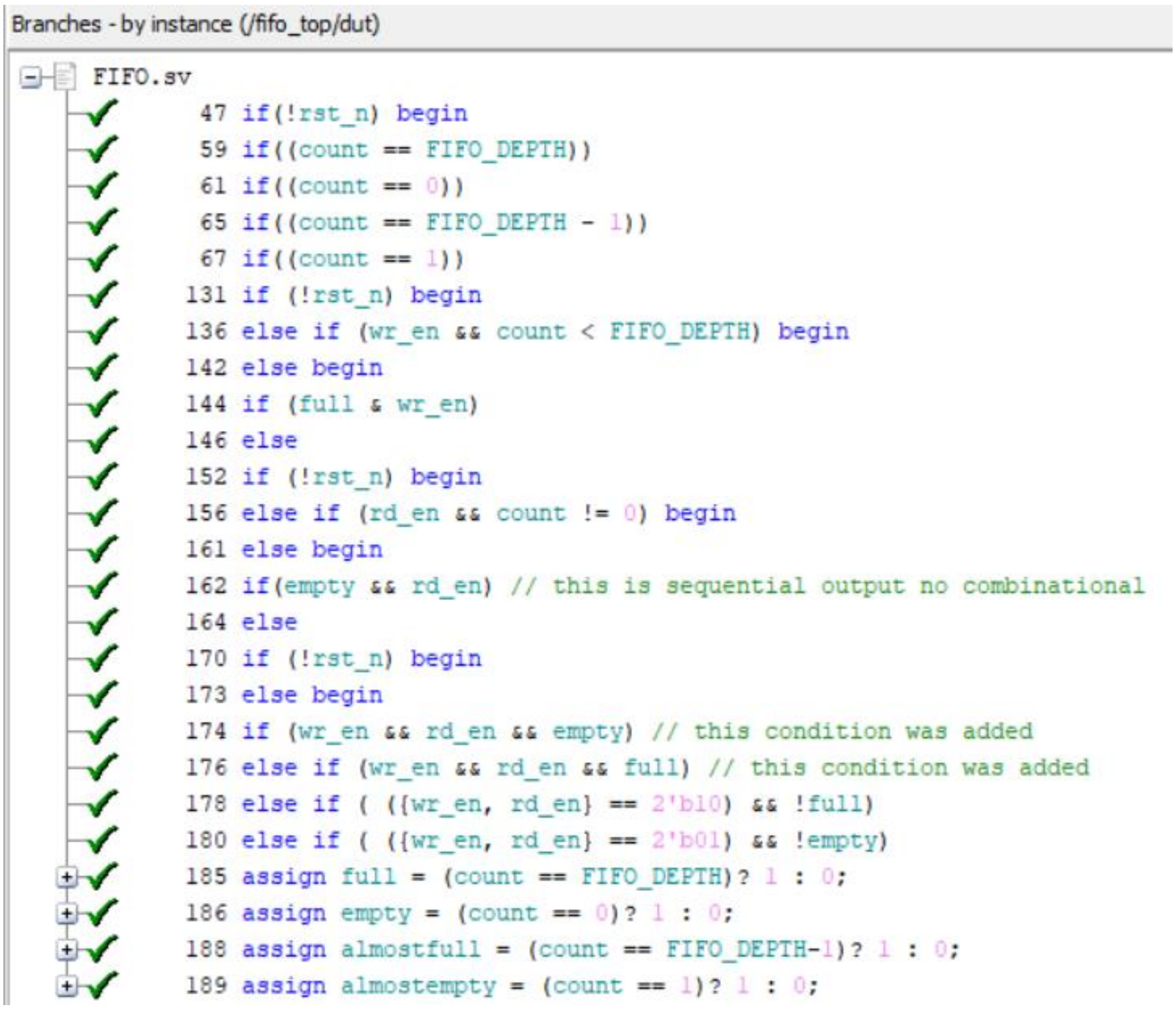
Name	Assertion Typ	Language	Enable	Failure Cou	Pass Count	Active Cou	Memor	P	Peak	Cu	ATV	Assertion Expression	Ind
/fifo_top/dut/count_aa	Immediate	SVA	on	0	1	-	-	..	-		off	assert (count==0)	✓
/fifo_top/dut/wr_ptr_aa	Immediate	SVA	on	0	1	-	-	..	-		off	assert (wr_ptr==0)	✓
/fifo_top/dut/rd_ptr_aa	Immediate	SVA	on	0	1	-	-	..	-		off	assert (rd_ptr==0)	✓
/fifo_top/dut/full_aa	Immediate	SVA	on	0	1	-	-	..	-		off	assert (full==0)	✓
/fifo_top/dut/empty_aa	Immediate	SVA	on	0	1	-	-	..	-		off	assert (empty==1)	✓
/fifo_top/dut/underflow_aa	Immediate	SVA	on	0	1	-	-	..	-		off	assert (underflow==0)	✓
/fifo_top/dut/almostfull_aa	Immediate	SVA	on	0	1	-	-	..	-		off	assert (almostfull==0)	✓
/fifo_top/dut/almostempty_aa	Immediate	SVA	on	0	1	-	-	..	-		off	assert (almostempty==0)	✓
/fifo_top/dut/wr_ack_aa	Immediate	SVA	on	0	1	-	-	..	-		off	assert (wr_ack==0)	✓
/fifo_top/dut/overflow_aa	Immediate	SVA	on	0	1	-	-	..	-		off	assert (overflow==0)	✓
/fifo_top/dut/full_a	Immediate	SVA	on	0	1	-	-	..	-		off	assert (full==1)	✓
/fifo_top/dut/empty_a	Immediate	SVA	on	0	1	-	-	..	-		off	assert (empty==1)	✓
/fifo_top/dut/almostfull_a	Immediate	SVA	on	0	1	-	-	..	-		off	assert (almostfull==1)	✓
/fifo_top/dut/almostempty_a	Immediate	SVA	on	0	1	-	-	..	-		off	assert (almostempty==1)	✓
/fifo_top/dut/ack_a	Concurrent	SVA	on	0	1	-	0B...	..	...		off	assert(@(posedge clk) disable iff (rst_n==0) (((wr_en&&count<8))|=>wr_ack))	✓
/fifo_top/dut/overflow_a	Concurrent	SVA	on	0	1	-	0B...	..	...		off	assert(@(posedge clk) disable iff (rst_n==0) (((count==8&&wr_en))|=>overflow==1))	✓
/fifo_top/dut/underflow_a	Concurrent	SVA	on	0	1	-	0B...	..	...		off	assert(@(posedge clk) disable iff (rst_n==0) (((empty&&rd_en))|=>underflow))	✓
/fifo_top/dut/wr_ptr_a	Concurrent	SVA	on	0	1	-	0B...	..	...		off	assert(@(posedge clk) disable iff (rst_n==0) (((wr_en&&count<8))|=>wr_ptr==$past(wr_ptr)+1))	✓
/fifo_top/dut/rd_ptr_a	Concurrent	SVA	on	0	1	-	0B...	..	...		off	assert(@(posedge clk) disable iff (rst_n==0) (((rd_en&&count!=0))|=>rd_ptr==$past(rd_ptr)+1))	✓
/fifo_top/dut/count_write_priority_a	Concurrent	SVA	on	0	1	-	0B...	..	...		off	assert(@(posedge clk) disable iff (rst_n==0) ((((wr_en&&rd_en)&&empty))|=>count==$past(count)+1))	✓
/fifo_top/dut/count_read_priority_a	Concurrent	SVA	on	0	1	-	0B...	..	...		off	assert(@(posedge clk) disable iff (rst_n==0) ((((wr_en&&rd_en)&&full))|=>count==$past(count)-1))	✓
/fifo_top/dut/count_w_a	Concurrent	SVA	on	0	1	-	0B...	..	...		off	assert(@(posedge clk) disable iff (rst_n==0) ((((wr_en&&~rd_en)&&~full))|=>count==$past(count)+1))	✓
/fifo_top/dut/count_r_a	Concurrent	SVA	on	0	1	-	0B...	..	...		off	assert(@(posedge clk) disable iff (rst_n==0) ((((~wr_en&&rd_en)&&~empty))|=>count==$past(count)-1))	✓
/fifo_top/tb/#ublk#217929410#38/immed__39	Immediate	SVA	on	0	1	-	-	..	-		off	assert (randomize(...))	✓



/fifo_top/dut/count_aa	INACTIVE
/fifo_top/dut/wr_ptr_aa	INACTIVE
/fifo_top/dut/rd_ptr_aa	INACTIVE
/fifo_top/dut/full_aa	INACTIVE
/fifo_top/dut/empty_aa	INACTIVE
/fifo_top/dut/underflow_aa	INACTIVE
/fifo_top/dut/almostfull_aa	INACTIVE
/fifo_top/dut/almostempty_aa	INACTIVE
/fifo_top/dut/wr_ack_aa	INACTIVE
/fifo_top/dut/overflow_aa	INACTIVE
/fifo_top/dut/full_a	INACTIVE
/fifo_top/dut/empty_a	INACTIVE
/fifo_top/dut/almostfull_a	INACTIVE
/fifo_top/dut/almostempty_a	INACTIVE
/fifo_top/dut/ack_a	ACTIVE
/fifo_top/dut/overflow_a	INACTIVE
/fifo_top/dut/underflow_a	INACTIVE
/fifo_top/dut/wr_ptr_a	ACTIVE
/fifo_top/dut/rd_ptr_a	ACTIVE
/fifo_top/dut/count_write_priority_a	INACTIVE
/fifo_top/dut/count_read_priority_a	INACTIVE
/fifo_top/dut/count_w_a	INACTIVE
/fifo_top/dut/count_r_a	INACTIVE

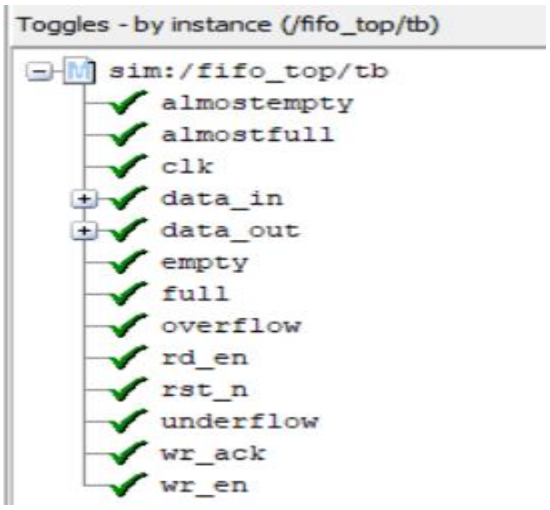
Code Coverage:

- Branch:



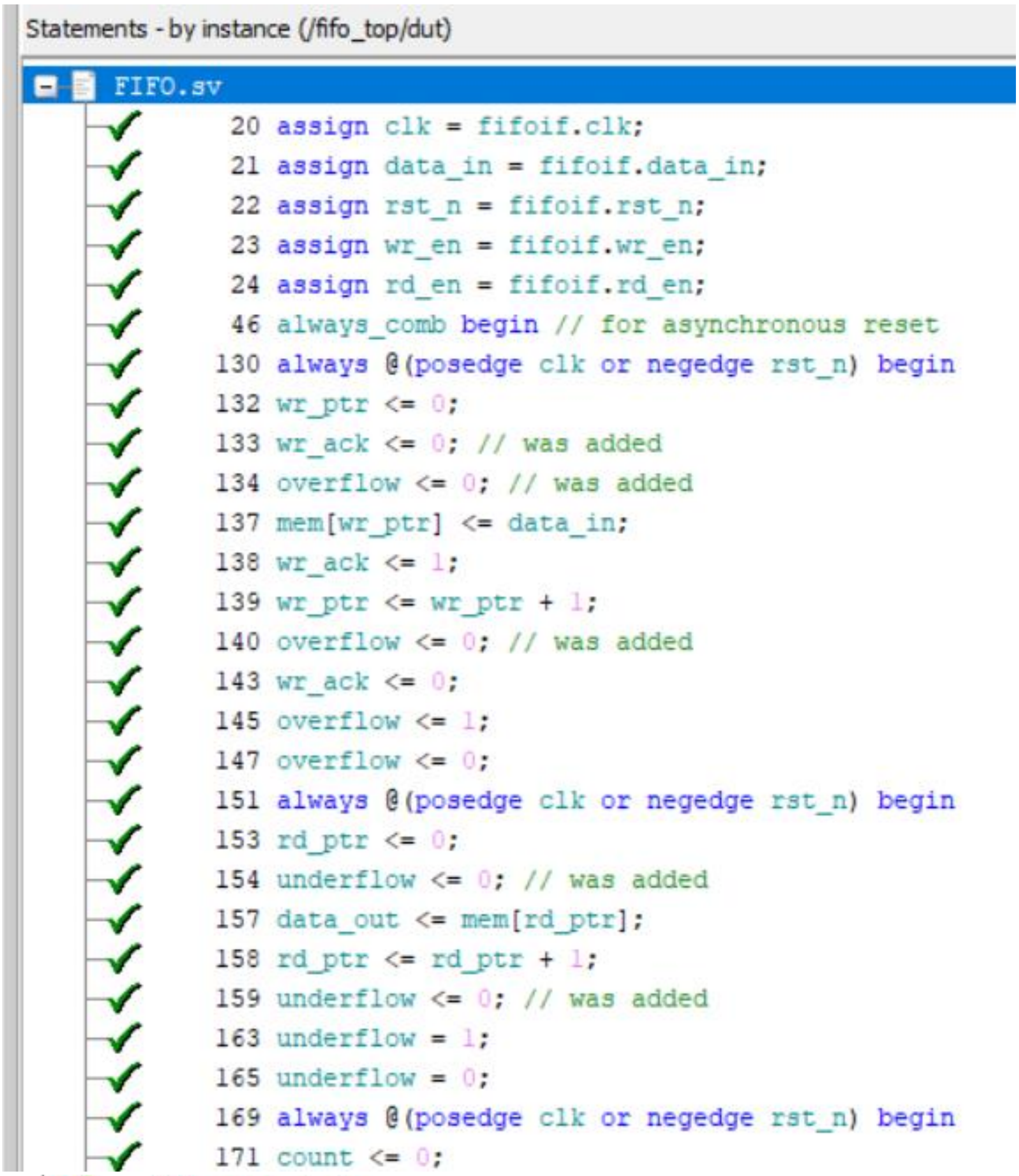
```
Branches - by instance (/fifo_top/dut)
FIFO.sv
✓  47 if(!rst_n) begin
✓  59 if((count == FIFO_DEPTH))
✓  61 if((count == 0))
✓  65 if((count == FIFO_DEPTH - 1))
✓  67 if((count == 1))
✓ 131 if (!rst_n) begin
✓ 136 else if (wr_en && count < FIFO_DEPTH) begin
✓ 142 else begin
✓ 144 if (full & wr_en)
✓ 146 else
✓ 152 if (!rst_n) begin
✓ 156 else if (rd_en && count != 0) begin
✓ 161 else begin
✓ 162 if(empty && rd_en) // this is sequential output no combinational
✓ 164 else
✓ 170 if (!rst_n) begin
✓ 173 else begin
✓ 174 if (wr_en && rd_en && empty) // this condition was added
✓ 176 else if (wr_en && rd_en && full) // this condition was added
✓ 178 else if ( ({wr_en, rd_en} == 2'b10) && !full)
✓ 180 else if ( ({wr_en, rd_en} == 2'b01) && !empty)
✓ 185 assign full = (count == FIFO_DEPTH)? 1 : 0;
✓ 186 assign empty = (count == 0)? 1 : 0;
✓ 188 assign almostfull = (count == FIFO_DEPTH-1)? 1 : 0;
✓ 189 assign almostempty = (count == 1)? 1 : 0;
```

- Toggle:



```
Toggles - by instance (/fifo_top/tb)
sim:/fifo_top/tb
✓ almostempty
✓ almostfull
✓ clk
✓ data_in
✓ data_out
✓ empty
✓ full
✓ overflow
✓ rd_en
✓ rst_n
✓ underflow
✓ wr_ack
✓ wr_en
```

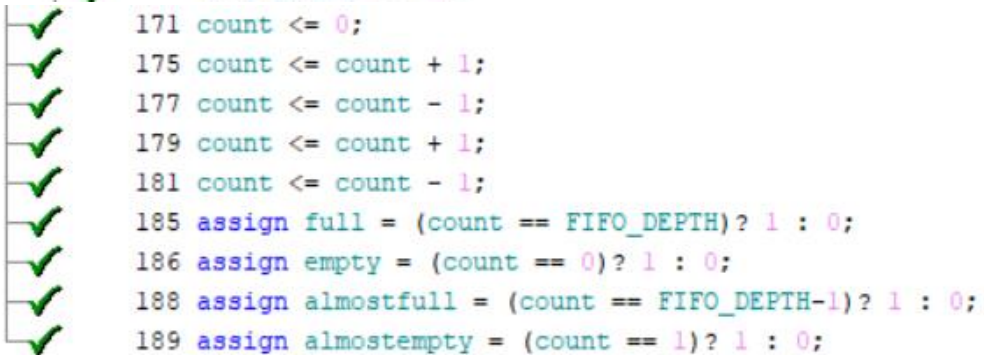
- Statement:



Statements - by instance (/fifo_top/dut)

FIFO.sv

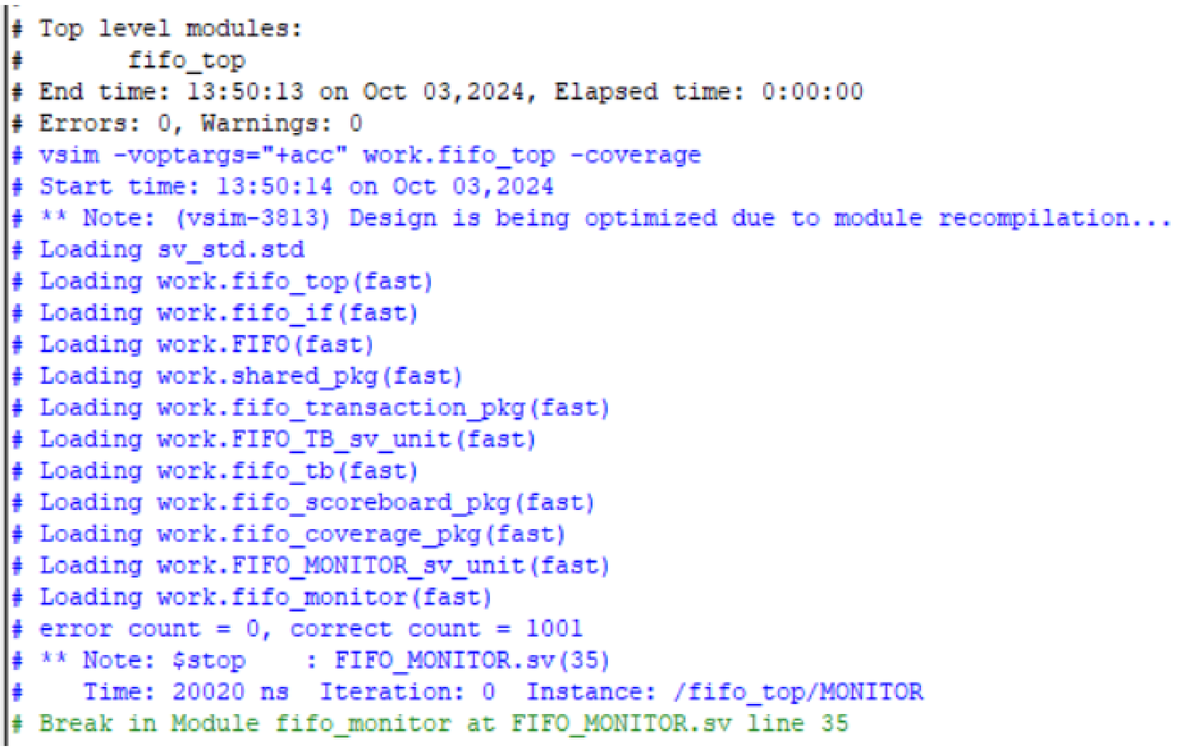
```
✓ assign clk = fifoif.clk;
✓ assign data_in = fifoif.data_in;
✓ assign rst_n = fifoif.rst_n;
✓ assign wr_en = fifoif.wr_en;
✓ assign rd_en = fifoif.rd_en;
✓ always_comb begin // for asynchronous reset
✓ always @(posedge clk or negedge rst_n) begin
✓ wr_ptr <= 0;
✓ wr_ack <= 0; // was added
✓ overflow <= 0; // was added
✓ mem[wr_ptr] <= data_in;
✓ wr_ack <= 1;
✓ wr_ptr <= wr_ptr + 1;
✓ overflow <= 0; // was added
✓ wr_ack <= 0;
✓ overflow <= 1;
✓ overflow <= 0;
✓ always @(posedge clk or negedge rst_n) begin
✓ rd_ptr <= 0;
✓ underflow <= 0; // was added
✓ data_out <= mem[rd_ptr];
✓ rd_ptr <= rd_ptr + 1;
✓ underflow <= 0; // was added
✓ underflow = 1;
✓ underflow = 0;
✓ always @(posedge clk or negedge rst_n) begin
✓ count <= 0;
```



```
✓ count <= 0;
✓ count <= count + 1;
✓ count <= count - 1;
✓ count <= count + 1;
✓ count <= count - 1;
✓ assign full = (count == FIFO_DEPTH)? 1 : 0;
✓ assign empty = (count == 0)? 1 : 0;
✓ assign almostfull = (count == FIFO_DEPTH-1)? 1 : 0;
✓ assign almostempty = (count == 1)? 1 : 0;
```

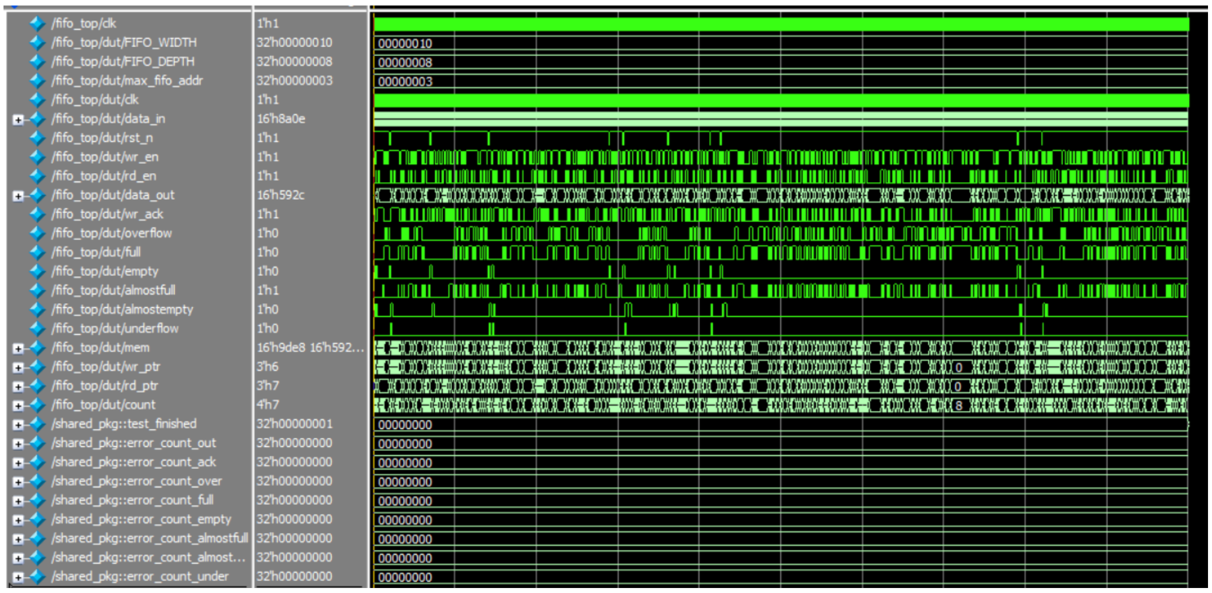
Snippets:

- Error count never executed:



```
# Top level modules:
#       fifo_top
# End time: 13:50:13 on Oct 03,2024, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# vsim -voptargs="+acc" work.fifo_top -coverage
# Start time: 13:50:14 on Oct 03,2024
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
# Loading sv_std.std
# Loading work.fifo_top(fast)
# Loading work.fifo_if(fast)
# Loading work.FIFO(fast)
# Loading work.shared_pkg(fast)
# Loading work.fifo_transaction_pkg(fast)
# Loading work.FIFO_TB_sv_unit(fast)
# Loading work.fifo_tb(fast)
# Loading work.fifo_scoreboard_pkg(fast)
# Loading work.fifo_coverage_pkg(fast)
# Loading work.FIFO_MONITOR_sv_unit(fast)
# Loading work.fifo_monitor(fast)
# error count = 0, correct count = 1001
# ** Note: $stop    : FIFO_MONITOR.sv(35)
#    Time: 20020 ns  Iteration: 0  Instance: /fifo_top/MONITOR
# Break in Module fifo_monitor at FIFO_MONITOR.sv line 35
```

- Full wave:



```
/fifo_top/clk                          1'h1
/fifo_top/dut/FIFO_WIDTH               32'h00000010
/fifo_top/dut/FIFO_DEPTH               32'h00000008
/fifo_top/dut/max_fifo_addr            32'h00000003
/fifo_top/dut/clk                      1'h1
/fifo_top/dut/data_in                  16'h8a0e
/fifo_top/dut/rst_n                    1'h1
/fifo_top/dut/wr_en                    1'h1
/fifo_top/dut/rd_en                    1'h1
/fifo_top/dut/data_out                 16'h592c
/fifo_top/dut/wr_ack                   1'h1
/fifo_top/dut/overflow                 1'h0
/fifo_top/dut/full                     1'h0
/fifo_top/dut/empty                    1'h0
/fifo_top/dut/almostfull               1'h1
/fifo_top/dut/almostempty              1'h0
/fifo_top/dut/underflow                1'h0
/fifo_top/dut/mem                      16'h9de8 16'h592...
/fifo_top/dut/wr_ptr                   3'h6
/fifo_top/dut/rd_ptr                   3'h7
/fifo_top/dut/count                    4'h7
/shared_pkg::test_finished             32'h00000001
/shared_pkg::error_count_out           32'h00000000
/shared_pkg::error_count_ack           32'h00000000
/shared_pkg::error_count_over          32'h00000000
/shared_pkg::error_count_full          32'h00000000
/shared_pkg::error_count_empty         32'h00000000
/shared_pkg::error_count_almostfull    32'h00000000
/shared_pkg::error_count_almost...     32'h00000000
/shared_pkg::error_count_under         32'h00000000
```