**Synchronous FIFO – UVM**

**Amira Atef Ismaeil El Komy**

Table of Contents

[1. Verification Plan 4](#_Toc203325424)

[2. UVM Testbench Structure 5](#_Toc203325425)

[3. Bugs Found 7](#_Toc203325426)

[4. Source Files List 8](#_Toc203325427)

[5. Do File 8](#_Toc203325428)

[6. Coverage Report Snippets 8](#_Toc203325429)

[5.1. Branch Coverage Snippet 8](#_Toc203325434)

[5.2. Toggle Coverage Snippet 9](#_Toc203325435)

[5.3. Statement Coverage Snippet 9](#_Toc203325436)

[5.4. Condition Coverage Snippet 9](#_Toc203325437)

[7. Functional Coverage Report Snippet 10](#_Toc203325438)

[8. Assertions Report Snippet 10](#_Toc203325439)

[9. Cover Directives Snippet 11](#_Toc203325440)

[10. Waveform Snippets 12](#_Toc203325441)

[11. Transcript Snippets 13](#_Toc203325442)

List of Figures

[Figure 1: Sync. FIFO Verification Plan 4](#_Toc203325443)

[Figure 2: FIFO UVM Environment Architecture 5](#_Toc203325444)

[Figure 3: Bugs Found 1 7](#_Toc203325445)

[Figure 4: Bugs Found 2 7](#_Toc203325446)

[Figure 5: Bugs Found 3 7](#_Toc203325447)

[Figure 6: Bugs Found 4 7](#_Toc203325448)

[Figure 7: Bugs Found 5 7](#_Toc203325449)

[Figure 8: List of Source Files 8](#_Toc203325450)

[Figure 9: The Do File 8](#_Toc203325451)

[Figure 10: Total Branch Coverage 8](#_Toc203325452)

[Figure 11: Total Toggle Coverage 9](#_Toc203325453)

[Figure 12: Total Statement Coverage 9](#_Toc203325454)

[Figure 13: Total Condition Coverage 9](#_Toc203325455)

[Figure 14: Total Functional Coverage 10](#_Toc203325456)

[Figure 15: Assertions 10](#_Toc203325457)

[Figure 16: Cover Directives 11](#_Toc203325458)

[Figure 17: Reset Sequence Waveform 12](#_Toc203325459)

[Figure 18: Write Only Sequence Waveform 12](#_Toc203325460)

[Figure 19: Read Only Sequence Waveform 12](#_Toc203325461)

[Figure 20: Write + Read Sequence Waveform 12](#_Toc203325462)

[Figure 21: Transcript Snippet 13](#_Toc203325463)

[Figure 22: Transcript Snippet (Cont.) 13](#_Toc203325464)

List of Tables

[Table 1: Assertions 11](#_Toc203325465)

# Verification Plan

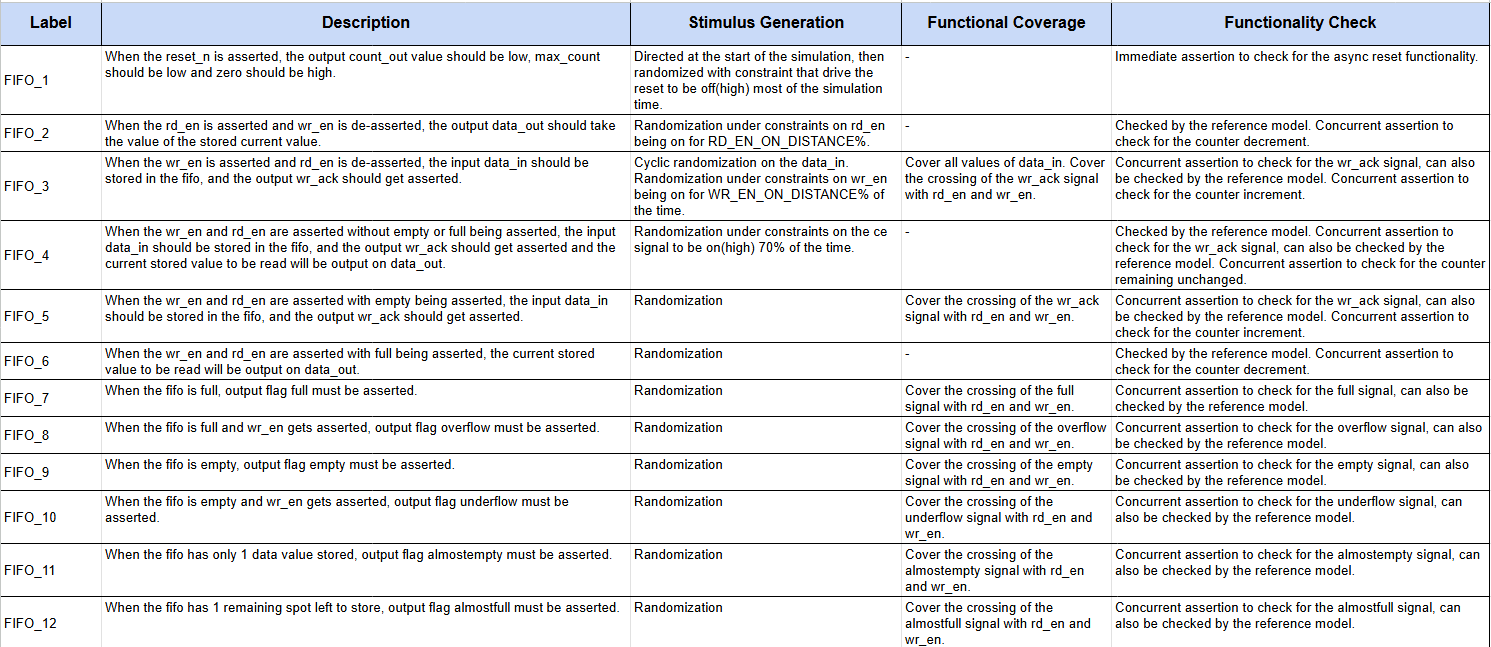


Figure 1: Sync. FIFO Verification Plan

# UVM Testbench Structure

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2: FIFO UVM Environment Architecture

The UVM testbench flow:

* The top module generates the clock and instantiates the test, interface and FIFO, our design under test.
* The interface supplies the DUT with the input stimulus and receives the output from it, so the interface carries all the signals.
* The test class, which extends uvm\_test, instantiates all sequence classes and the environment. It also runs all the sequences.
* The sequence item class, which extends uvm\_seq\_item, contains all the variables for all the signals passing through the environment whether these signals are DUT input signals that get randomized or not. In addition, it contains all constraint blocks for randomized signals.
* The sequence class, which extends uvm\_sequence, is a base class that contains tasks for all different sequences, which are then called in child classes of the base class to run different sequences.
* The environment class, which extends uvm\_env, instantiates the coverage collector, scoreboard and the agent.
* The agent class, which extends uvm\_agent, instantiates the monitor, driver and sequencer, as well as creates the connections using uvm\_tlm\_ports and uvm\_analysis\_ports, between the monitor and classes outside the agent or between the driver and sequencer, respectively.
* The driver class, which extends uvm\_driver, instantiates a seq\_item, as it drives the input stimulus to the interface given to it by the sequence in a seq\_item through the sequencer.
* The monitor class, which extends uvm\_monitor, instantiates a seq\_item, and writes all the signals in the interface to a uvm\_fifo every negative edge clock.
* The coverage class, which extends uvm\_component, instantiates a seq\_item, to be able to get the sequence item from the monitor and applies coverpoints inside covergroups to its signals and samples the covergroups.
* The scoreboard class, which extends uvm\_scoreboard, instantiates a seq\_item, and compares every output using a reference model task written inside it, incrementing counters for every correct and incorrect comparison.
* An SVA class was also written containing all assertions and was bound to the DUT inside the top.

# Bugs Found

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3: Bugs Found 1

A close-up of a white background

AI-generated content may be incorrect.

Figure 4: Bugs Found 2

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5: Bugs Found 3

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6: Bugs Found 4

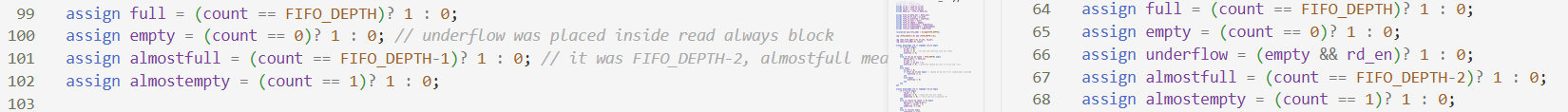


Figure 7: Bugs Found 5

# Source Files List

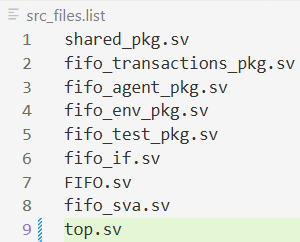


Figure 8: List of Source Files

# Do File

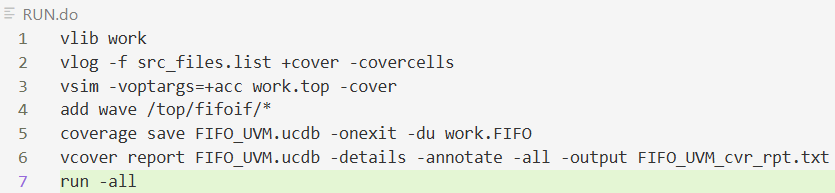


Figure 9: The Do File

# Coverage Report Snippets



## Branch Coverage Snippet

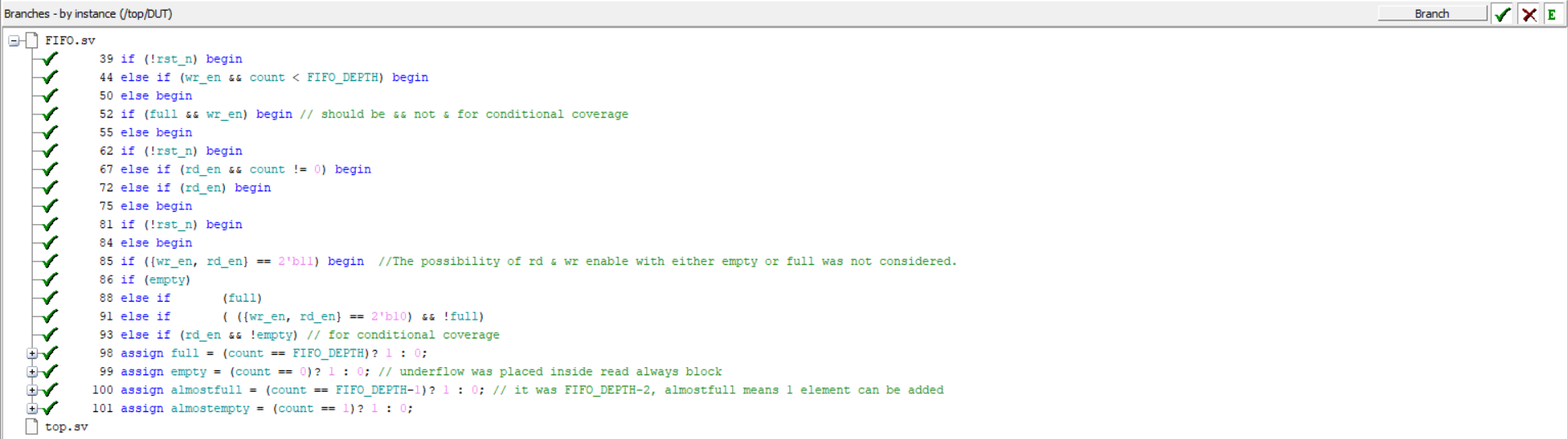


Figure 10: Total Branch Coverage

## Toggle Coverage Snippet

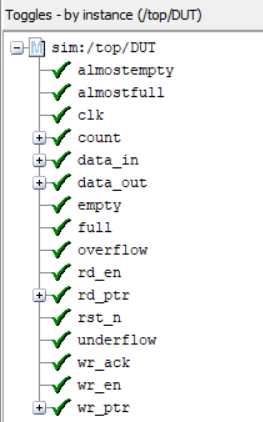


Figure 11: Total Toggle Coverage

## Statement Coverage Snippet

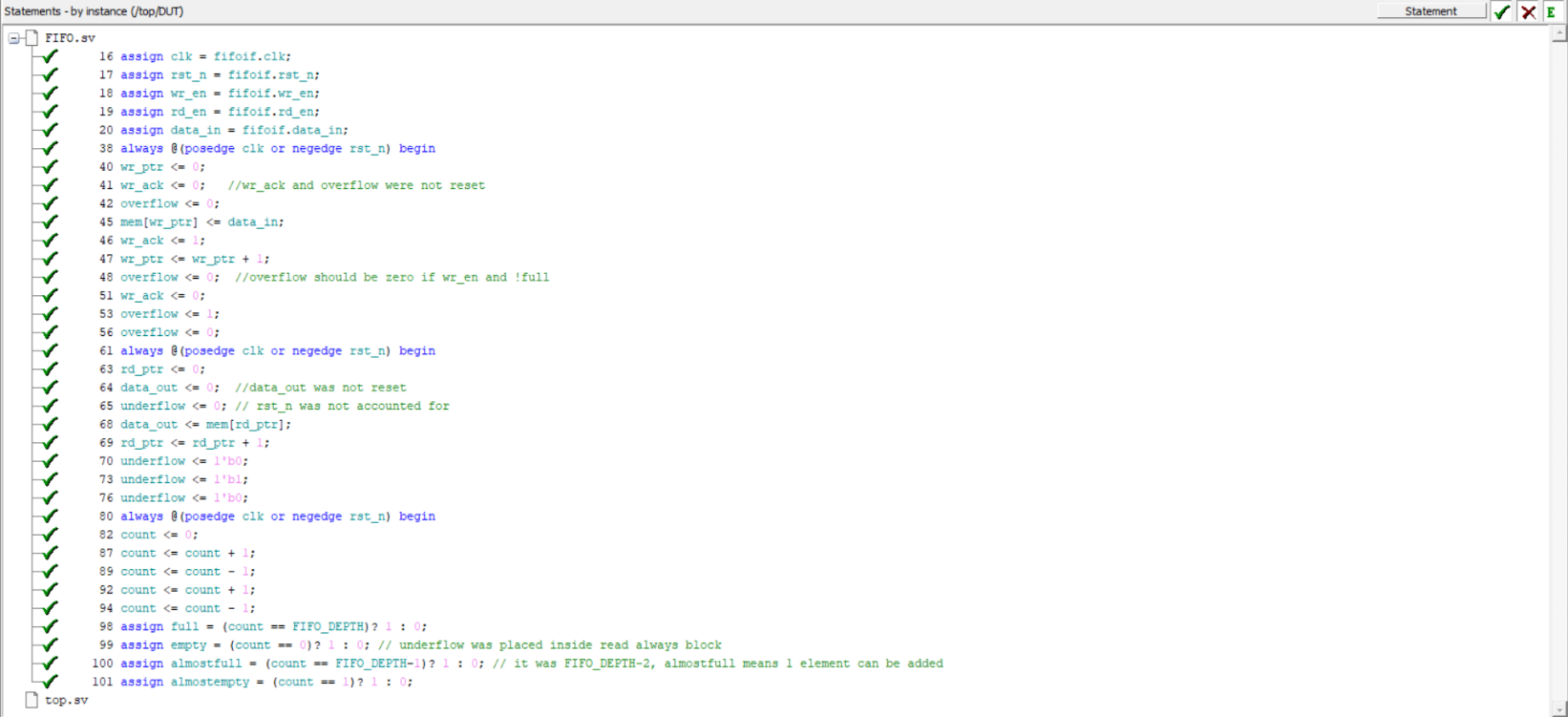


Figure 12: Total Statement Coverage

## Condition Coverage Snippet

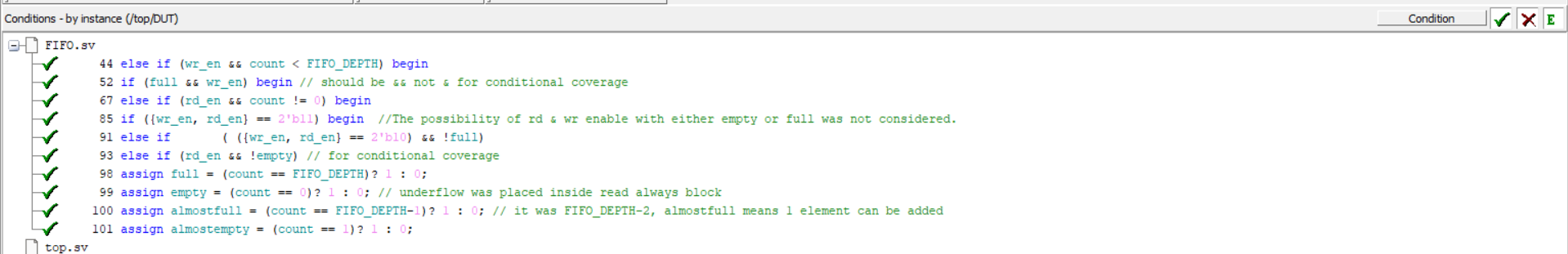


Figure 13: Total Condition Coverage

# Functional Coverage Report Snippet

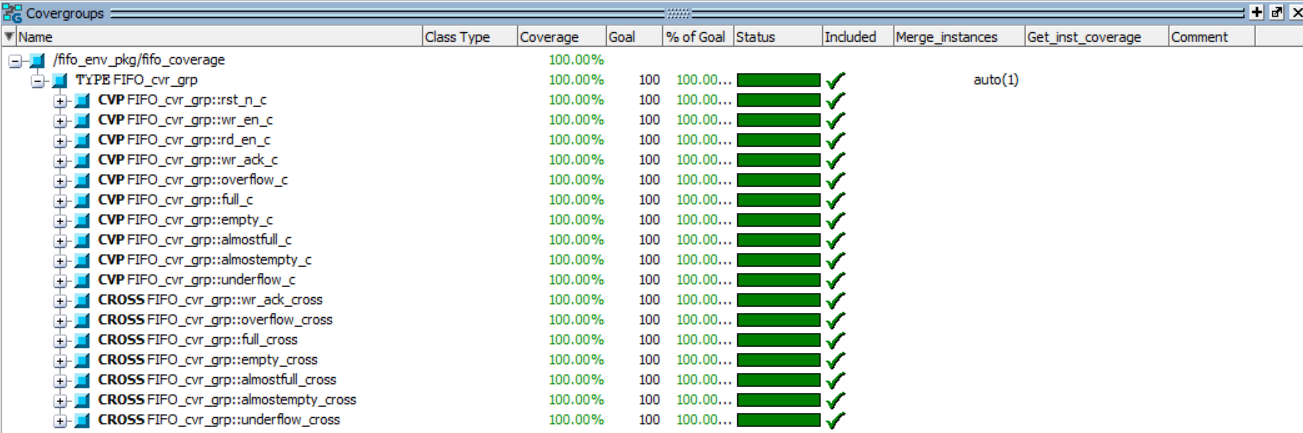


Figure 14: Total Functional Coverage

# Assertions Report Snippet

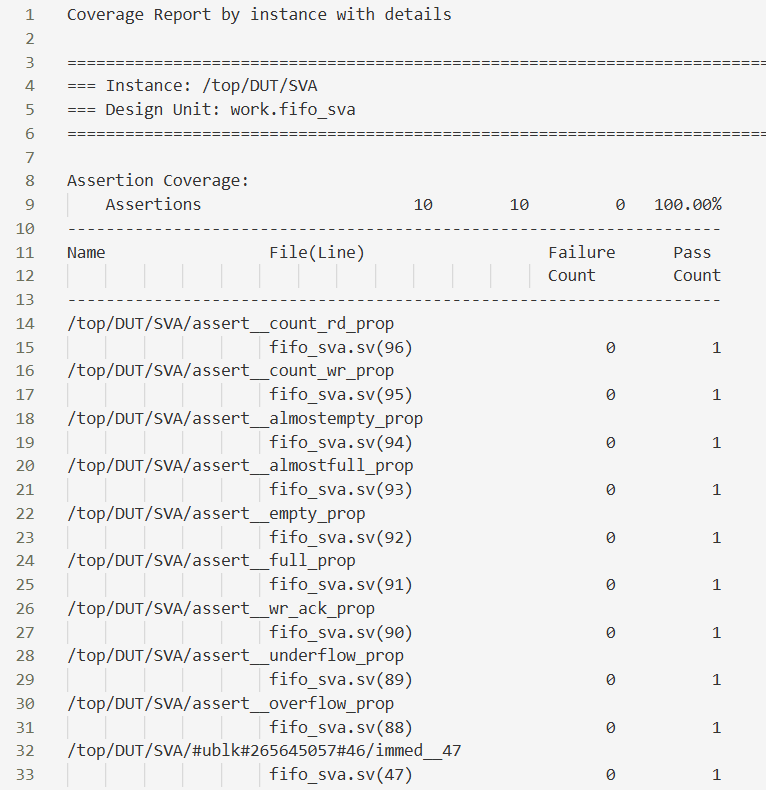


Figure 15: Assertions

Table 1: Assertions

|  |  |
| --- | --- |
| Feature | Assertion |
| When rst\_n is zero all outputs must be zero and only empty is 1. | always @(posedge clk) begin          if (!rst\_deasserted)  **assert** (data\_out == 0 && empty && !almostfull && !almostempty && !underflow && !overflow && !full && !wr\_ack && count == 0)      end |
| When FIFO is full and wr\_en gets asserted, overflow gets asserted. | @(posedge clk) disable iff (~rst\_n) (wr\_en && full) |=> overflow; |
| When FIFO is empty and rd\_en gets asserted, overflow gets asserted. | @(posedge clk) disable iff (~rst\_n) (rd\_en && empty) |=> underflow; |
| When FIFO is not full and wr\_en gets asserted, wr\_ack gets asserted. | @(posedge clk) disable iff (~rst\_n) (wr\_en && count < FIFO\_DEPTH) |=> wr\_ack; |
| When FIFO is full, as indicated by the counter, Full gets asserted. | @(posedge clk) disable iff (~rst\_n) (count == FIFO\_DEPTH) |-> full; |
| When FIFO is empty, as indicated by the counter, empty gets asserted. | @(posedge clk) disable iff (~rst\_n) (count == 0) |-> empty; |
| When only 1 place is empty in FIFO, almostfull gets asserted. | @(posedge clk) disable iff (~rst\_n) (count == FIFO\_DEPTH-1) |-> almostfull; |
| When only 1 place is full in FIFO, almostempty gets asserted. | @(posedge clk) disable iff (~rst\_n) (count == 1) |-> almostempty; |
| Counter must get incremented when wr\_en is asserted, fifo is not full and rd\_en is de-asserted or when wr\_en is asserted, fifo is empty and rd\_en is asserted. | @(posedge clk) disable iff (~rst\_n) ((wr\_en && !full && !rd\_en) || (wr\_en && rd\_en && empty)) |=> (count == **$past**(count) + 1); |
| Counter must get decremented when rd\_en is asserted, fifo is not empty and wr\_en is de-asserted or when rd\_en is asserted, fifo is full and wr\_en is asserted. | @(posedge clk) disable iff (~rst\_n) ((rd\_en && !empty && !wr\_en) || (wr\_en && rd\_en && full)) |=> (count == **$past**(count) - 1); |

# Cover Directives Snippet

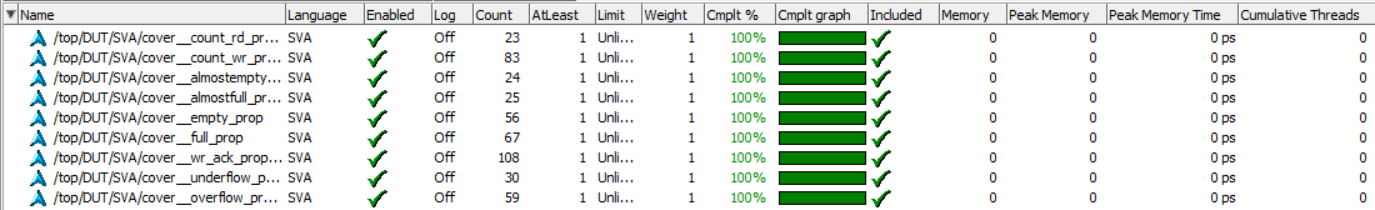


Figure 16: Cover Directives

# Waveform Snippets

A screenshot of a computer

AI-generated content may be incorrect.

Figure 17: Reset Sequence Waveform

A grid with green lines

AI-generated content may be incorrect.

Figure 18: Write Only Sequence Waveform

A screen shot of a graph

AI-generated content may be incorrect.

Figure 19: Read Only Sequence Waveform

A screenshot of a video game

AI-generated content may be incorrect.

Figure 20: Write + Read Sequence Waveform

# Transcript Snippets

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 21: Transcript Snippet

A white background with blue text

AI-generated content may be incorrect.

Figure 22: Transcript Snippet (Cont.)