ALU VERIFICATION

DOCUMENT

**NAME: Prathiksha**

**EMP ID: 6092**

**Mirafra Technologies Manipal**

**Table of contents**

**[CHAPTER 1:Project Overview and Specification](#_Toc6972)** [3](#_Toc6972)-4

[1.1 Project Overview: 3](#_Toc5178)

[1.2 Verification Objectives 3](#_Toc13295)

1.3 DUT Intefaces ..................................................................................................4

[2.1 ALU Testbench Architecture 5](#_Toc16510)

[2.2 Component details and Flowchart 5-6](#_Toc3360)

[ DUV 5](#_Toc9512)

[ Sequence item 5](#_Toc11961)

[ Sequence 5](#_Toc22679)

[ Sequencer 6](#_Toc7566)

[ Driver 6](#_Toc22239)

[ Monitor 6](#_Toc25210)

[ Scoreboard 6](#_Toc26739)

[ Coverage Collector 6](#_Toc11741)

[ Agent 6](#_Toc8846)

[ Environment 6](#_Toc21727)

[ Test 6](#_Toc12076)

[ Top module 6](#_Toc9530)

Virtual Interface 7

TLM............................................................................................................7

UVM Configuration Database ...................................................................7

**[CHAPTER 3:Verification Results](#_Toc27169)** [7](#_Toc27169)-10

[3.1 Errors in the DUT 7](#_Toc17510)

[3.2 Coverage Report 8](#_Toc8217)

[ Input Coverage 8](#_Toc23855)

[ Output coverage 8](#_Toc17997)

[ Assertion coverage 9](#_Toc14080)

[ Overall coverage 9](#_Toc31029)

[ Output waveform 1](#_Toc27740)0

**CHAPTER 1: PROJECT OVERVIEW AND SPECIFICATION**

## ****1 Project Overview****

This project focuses on the **verification of a parameterized Arithmetic Logic Unit (ALU)** using a **UVM-based verification environment**. The ALU, as defined in the design specification, supports a broad set of operations, including arithmetic, logical, comparison, and shift/rotate functions. Its **parameterized data width** adds flexibility but also increases the need to verify across multiple operand sizes.

The ALU accepts two operands (**OPA** and **OPB**), a carry-in (**CIN**), and a 4-bit command (**CMD**) that determines the operation to be performed. Additional control signals such as **CE**, **MODE** (arithmetic/logical), and **INP\_VALID** (operand validity) influence the ALU’s behavior, making it essential to verify all possible combinations.

The outputs of interest include the **result (RES)** and status flags — **COUT** (carry-out), **OFLOW** (overflow), **G, L, E** (comparison flags), and **ERR** (error indicator). Verification ensures that these outputs are correctly generated for all supported operations under valid, corner-case, and invalid conditions.

A **UVM testbench** has been developed to provide a structured, reusable, and scalable verification framework. The environment incorporates **stimulus generation (directed and constrained-random sequences), transaction-level communication, monitors, scoreboards, assertions, and functional coverage** to validate correctness against the reference model and achieve complete verification across all scenarios.

**1.2 Verification Objective**

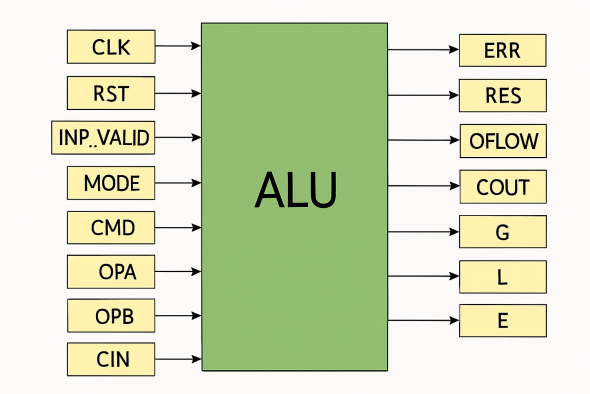
* **Functional Correctness:** Validate the RTL design by applying various valid, invalid, and corner-case input scenarios.
* **UVM Environment Development:** Build a modular and reusable testbench comprising UVM components such as sequences, sequencers, drivers, monitors, agents, environment, and scoreboard.
* **Operation Validation:** Verify that all supported ALU operations (arithmetic, logical, shift/rotate, comparison) produce expected results and status flags.
* **Reference Checking:** Compare DUT outputs against a golden reference model within the scoreboard to detect mismatches automatically.
* **Control Signal Verification:** Confirm correct DUT behavior under different combinations of **MODE, CMD, INP\_VALID, and CE** control signals.
* **Error Detection:** Ensure that the ERR flag is properly asserted in scenarios like invalid commands or operand timeouts.
* **Metric-Driven Verification:** Use **functional coverage, code coverage, and assertion coverage** to measure completeness and close verification goals.
* **Debug and Documentation:** Identify, analyze, and debug functional issues, while documenting the verification strategy, methodology, and results for traceability and project closure.

**1.3 DUT interfaces**

The alu\_if interface is used to connect the testbench components to the DUT. It defines all input and output signals required to drive and monitor the ALU. The interface includes signals like operands (OPA,

OPB), command (CMD), control inputs (MODE, CIN, CE, INP\_VALID), and output signals (RES, COUT, OFLOW, E, G, L, ERR). It also includes an active-high reset (RST) and a clock (clk).

OPB), command (CMD), control inputs (MODE, CIN, CE, INP\_VALID), and output signals (RES, COUT, OFLOW, E, G, L, ERR). It also includes an active-high reset (RST) and a clock (clk).





**Clocking blocks** are defined for different components in the testbench:

**drv\_cb** for the driver, which controls all input signals to the DUT.

**mon\_cb** for the monitor, which reads output signals from the DUT.

**Modports** are used to provide access to specific clocking blocks for each component:

**DRV** modport gives access to the driver clocking block.

**MON** modport gives access to the monitor clocking block.

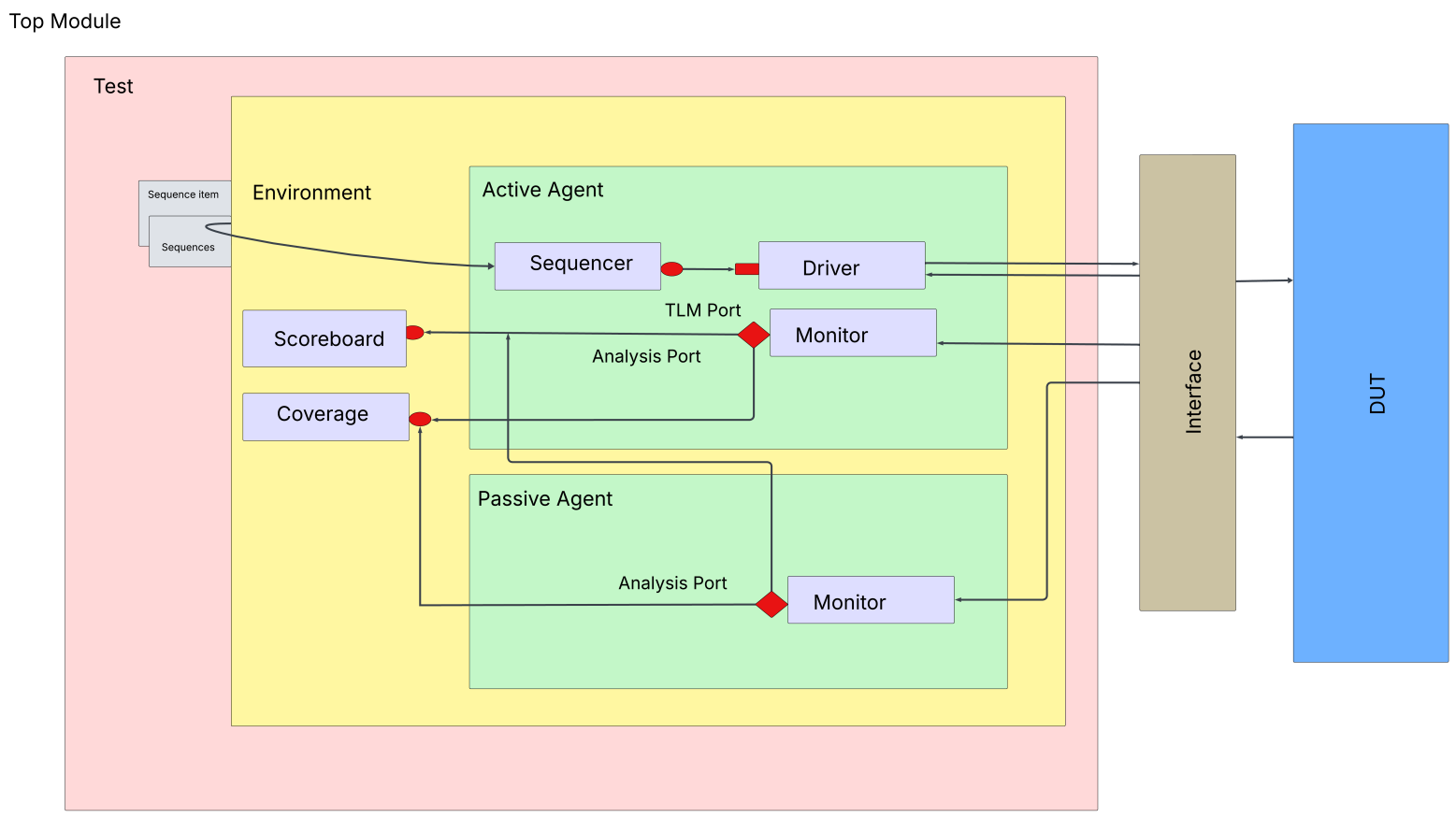
modport DRV (clocking drv\_cb, input clk, input RST);

modport MON (clocking mon\_cb, input clk, input RST);

This structure ensures synchronized and modular access to signals during verification.

**CHAPTER 2: TESTBENCH ARCHITECTURE AND METHODOLOGY**

**2.1. Testbench Architecture**

****

UVM (Universal Verification Methodology) is a SystemVerilog-based framework that provides a standardized, reusable, and scalable approach to functional verification. It follows a hierarchical, component-based structure built on standard base classes, communication mechanisms, and verification strategies. A typical UVM testbench includes key layers such as sequence items and sequences for stimulus, agents with drivers and monitors, scoreboards for result checking, and an environment that integrates and manages all components.

## 2.2 Component details

· **Design Under Verification (DUV):** The actual ALU RTL design being verified. It receives stimulus from the testbench and generates outputs that are later validated.

· **Sequence Item:** A transaction class that defines the data structure for one operation, including operands, control signals, and mode information.

· **Sequence:** A stimulus generator that creates and sends a stream of sequence items to the sequencer. It can be constrained-random or directed, depending on the test requirements.

· **Sequencer:** Acts as a scheduler between the sequence and the driver. It forwards sequence items to the driver via TLM communication, ensuring proper transaction flow.

· **Driver:** Converts sequence items into pin-level activities on the DUT interface. It drives valid signals and operands into the ALU.

· **Monitor:** Passively observes DUT outputs via the interface, transforms them back into transactions, and shares them with other components (like scoreboard and coverage) through analysis ports.

· **Scoreboard:** Performs result checking by comparing DUT outputs (from the monitor) with reference model outputs. Any differences are flagged as functional mismatches.

· **Coverage Collector:** Captures functional coverage, ensuring that various commands, modes, and corner cases are exercised during simulation. It connects to driver/monitor through TLM analysis ports.

· **Agent:** A reusable component that groups the sequencer, driver, and monitor into a single unit for easier instantiation and reusability.

· **Environment:** A higher-level container that brings together agents, scoreboards, coverage, and other utilities. It organizes the entire verification structure.

· **Test:** The top-level class in the UVM hierarchy. It configures the environment, starts sequences, and controls which scenarios are executed.

· **Top Module:** Instantiates the DUT, interface, and testbench environment. It also triggers the UVM phasing mechanism to kick off simulation.

**UVM Communication Mechanisms**

### · Virtual Interface: Provides access to DUT signals from UVM components. Set in the testbench top with uvm\_config\_db::set() and retrieved by components using uvm\_config\_db::get(). Drivers and monitors use it to drive or sample DUT signals.

· **TLM (Transaction Level Modeling):** Standardized ports/exports for component communication. Examples:

* uvm\_seq\_item\_pull\_port (sequencer ↔ driver)
* uvm\_analysis\_port (monitor → scoreboard/coverage)  
  Enables loose coupling and modular testbenches.

· **UVM Configuration Database:** Centralized mechanism to pass configs, objects, and virtual interfaces across components using a hierarchical path-based set/get approach.

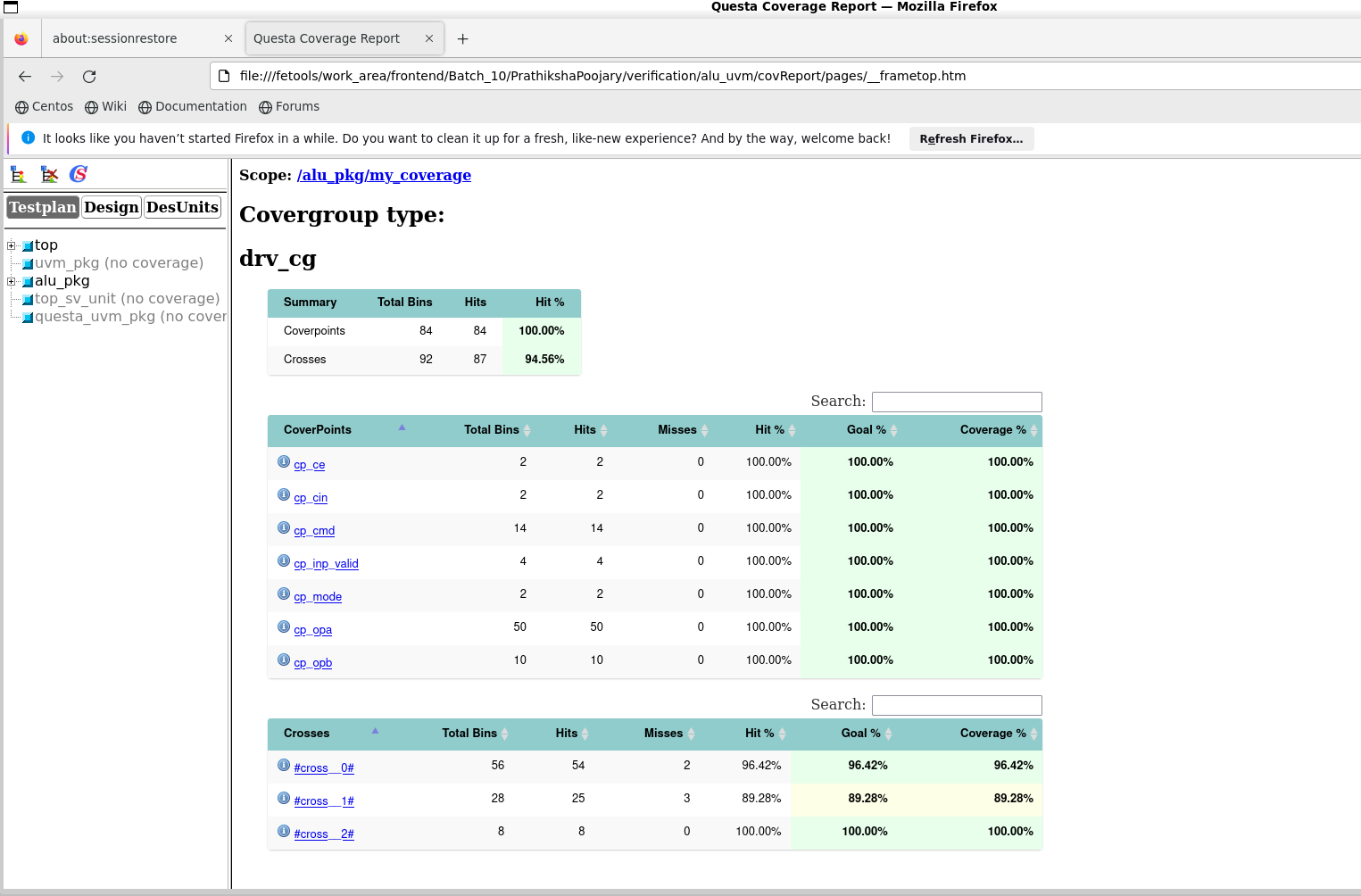
**CHAPTER 3: VERIFICATION RESULTS AND ANALYSIS**

**3.1: Error in the DUT**

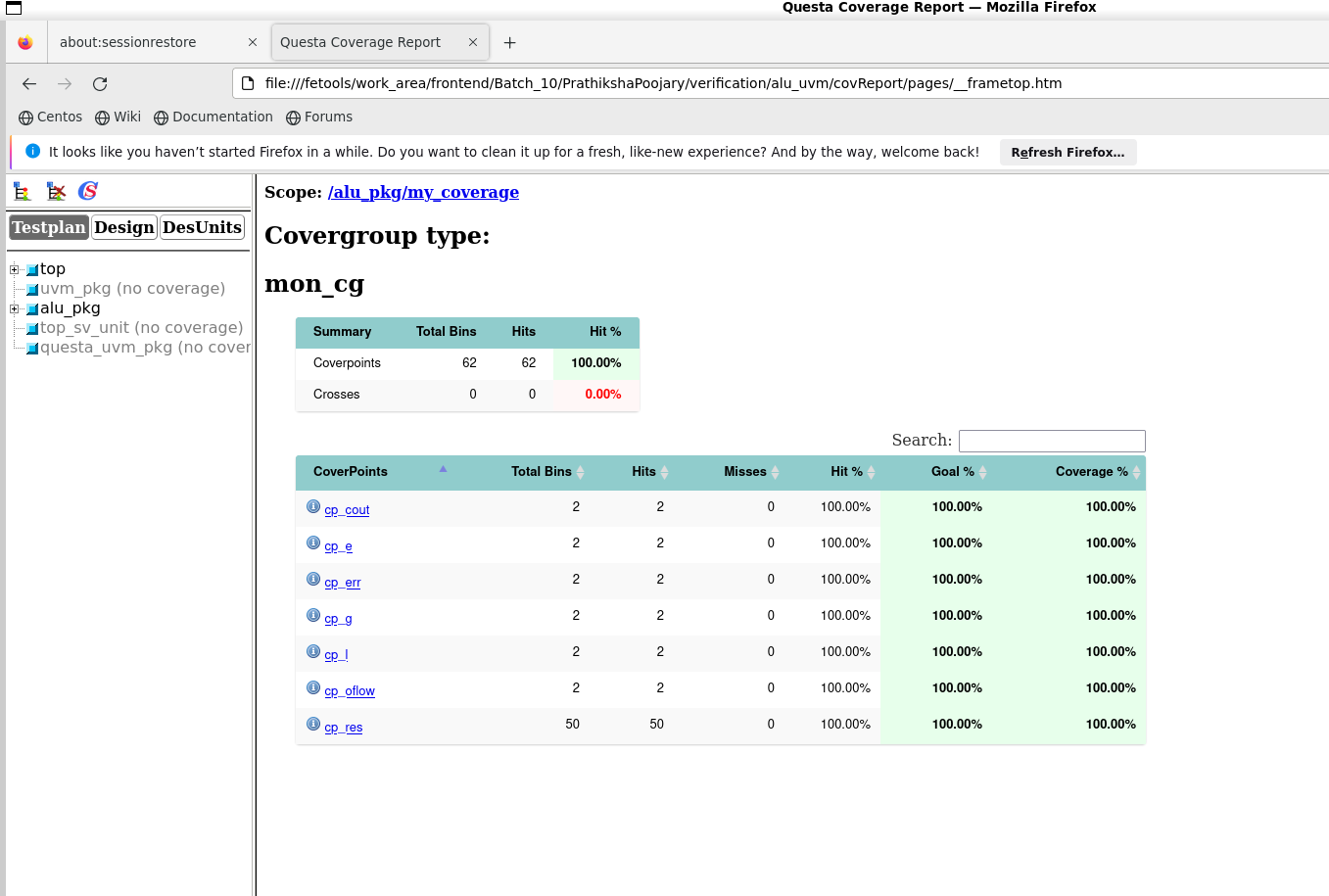
|  |  |
| --- | --- |
| Error | Description |
|  |  |
| ADD\_CIN | Carry-in functionality is not implemented correctly during addition. |
| SUB\_CIN | Carry-in functionality is not implemented correctly during subtraction. |
| INC\_B | Operand B is decremented instead of incremented. Also fails when INP\_VALID = 2'b01. |
| DEC\_B | Operand B is incremented instead of decremented. Also fails when INP\_VALID = 2'b01. |
| INC\_A | The operation fails when INP\_VALID = 2'b01. |
| DEC\_A | The operation fails when INP\_VALID = 2'b01. |
| OR | DUT performs logical AND instead of bitwise OR. |
| Shift and Multiply | Multiplication after shifting operand A is incorrect. |
| SHR1\_A | Right shift operation of operand A is incorrect. |
| SHR1\_B | DUT performs a left shift instead of right shift on operand B. |
| SHL1\_A | Works only when INP\_VALID = 2'b11; fails when INP\_VALID = 2'b01. |
| SHL1\_B | Works only when INP\_VALID = 2'b11; fails when INP\_VALID = 2'b01. |
| NOT\_A | Works only when INP\_VALID = 2'b11; fails when INP\_VALID = 2'b01. |
| NOT\_B | Works only when INP\_VALID = 2'b11; fails when INP\_VALID = 2'b01. |
| 16 Clock Cycle Waiting Period | DUT times out after 16 clock cycles due to missing error detection. |
| Error Handling | Error flag not asserted when CMD is out of range or when INP\_VALID = 2'b00. |
| Result Width | Output is limited to 9 bits, causing truncation and incorrect results, especially for multiplication. |

**3.2: Coverage Report**

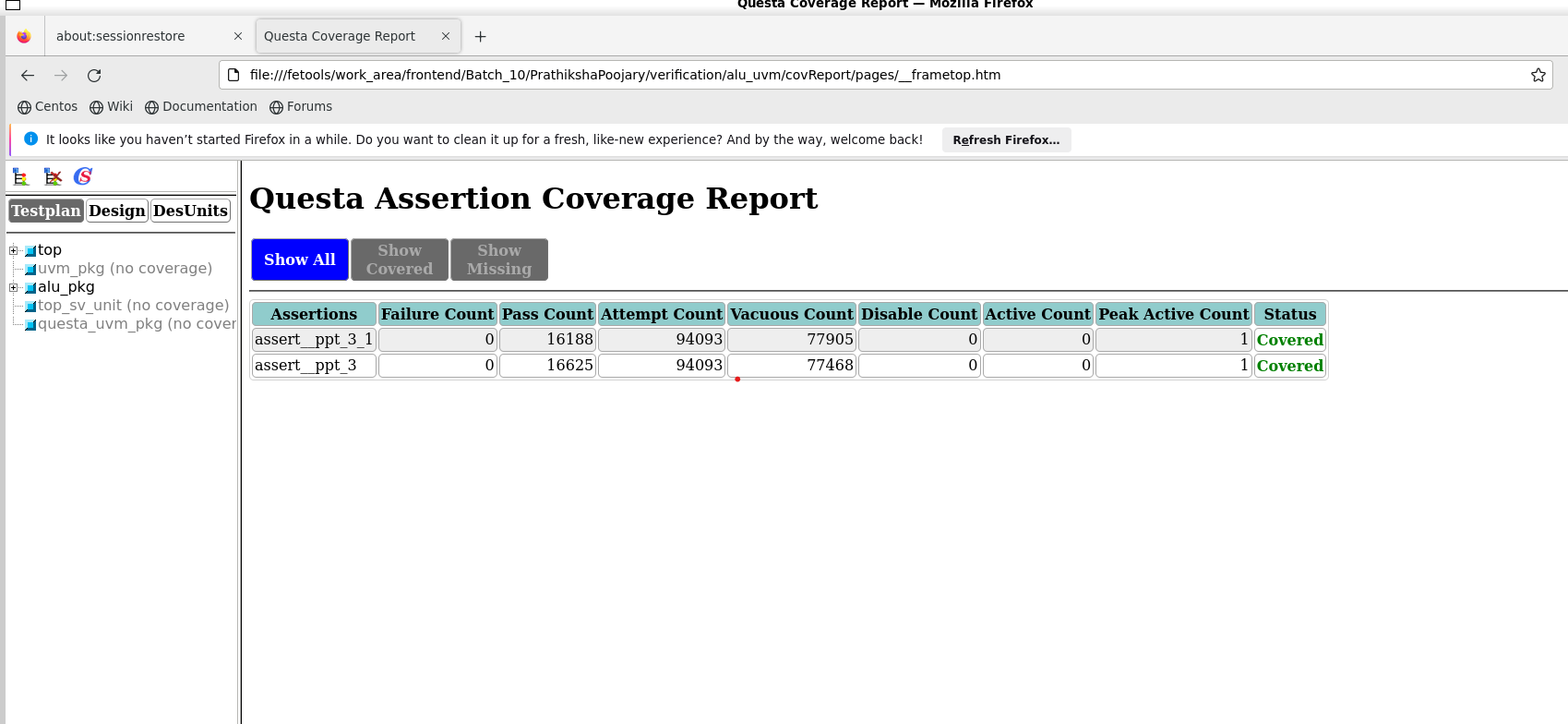
**3.2.1 Input Coverage**

****

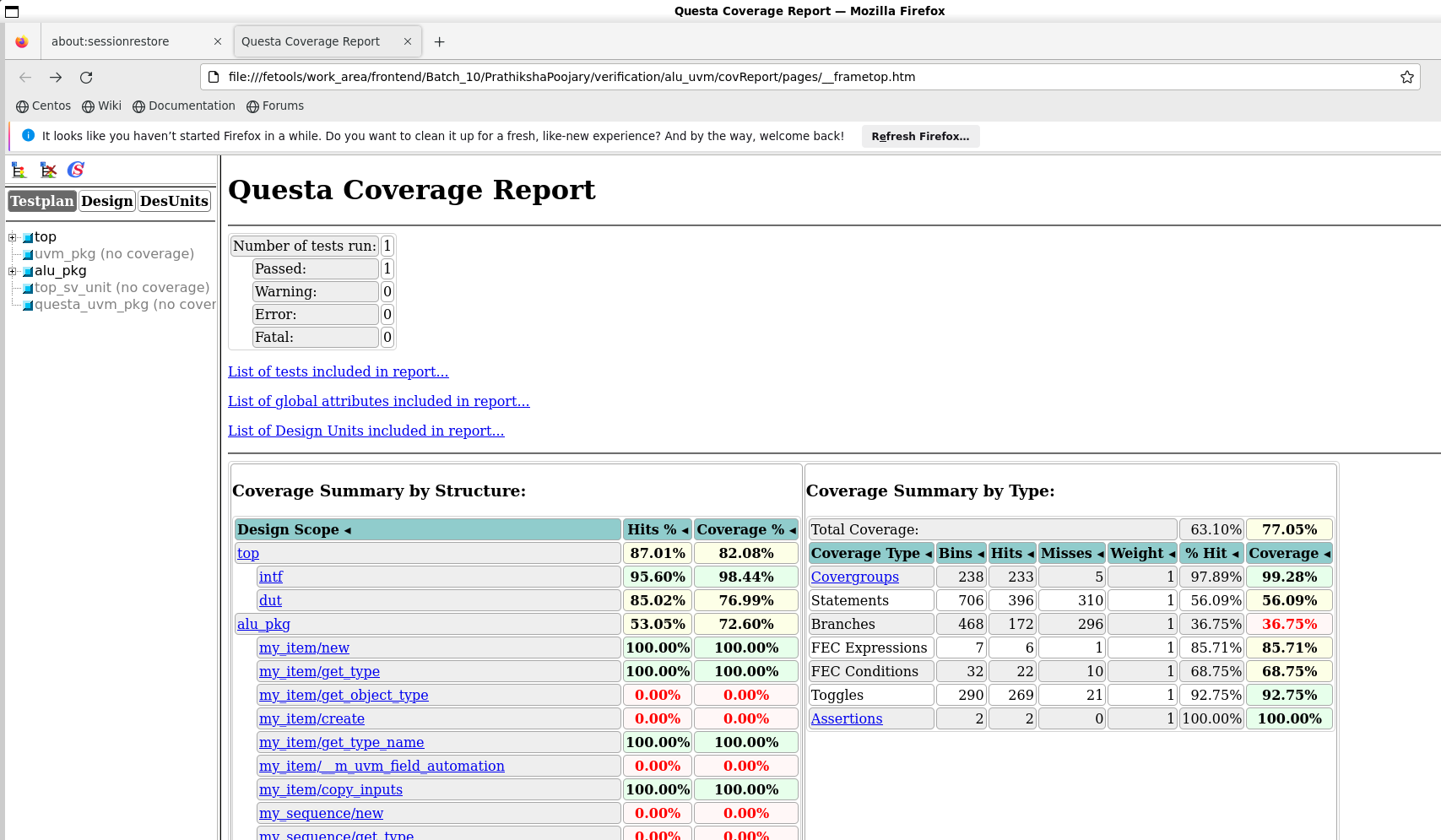
**3.2.2 Output Coverage**



**3.2.3 Assertion Coverage**



**3.2.4 Output Coverage**

****

**3.2.5 Output Waveform**

