Encoding IP Verification Plan

1. Introduction

1.1. Design Description

The Encoding module is a collection of 2 modules in the PCIe transmitter IP that have been grouped together for verification purposes. The position of the modules in the transmitter architecture is shown below.

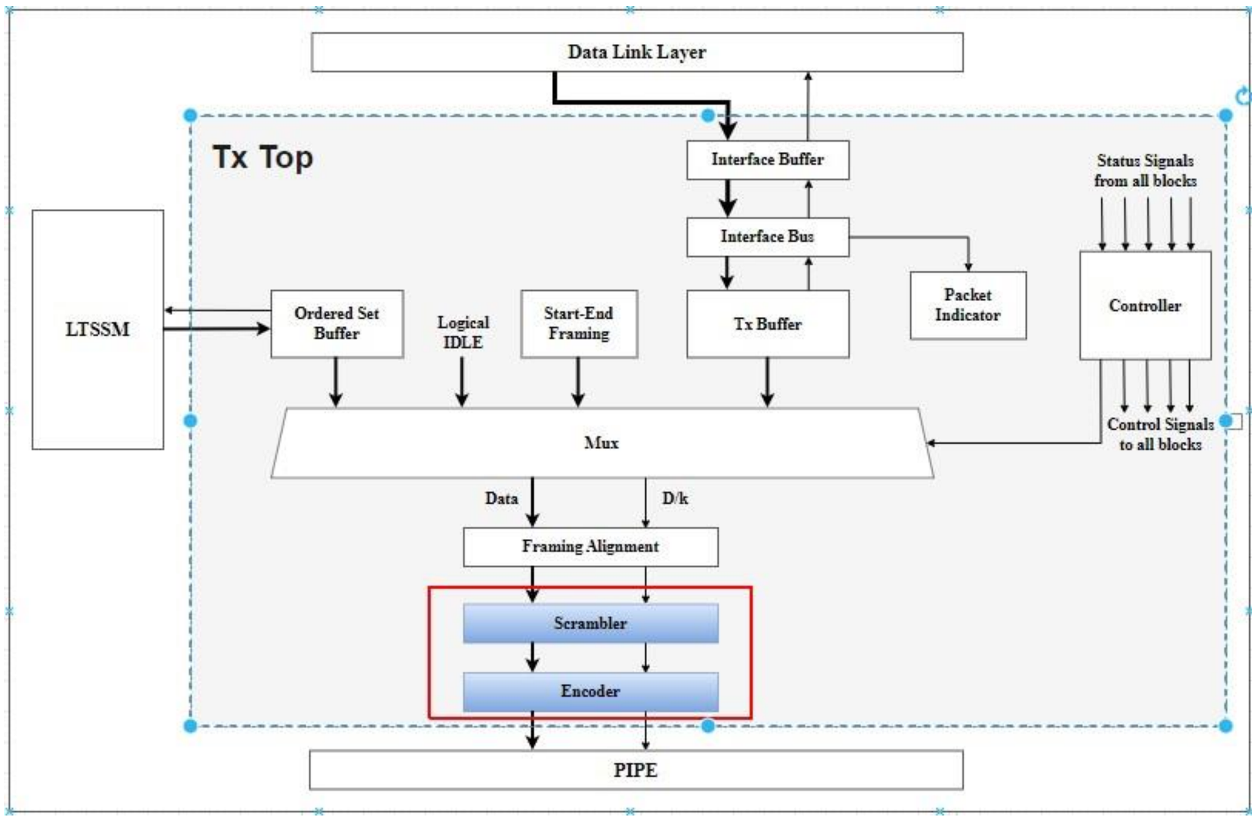


Figure 1: Position of the encoding modules in the transmitter architecture.

1.2. Block Diagram

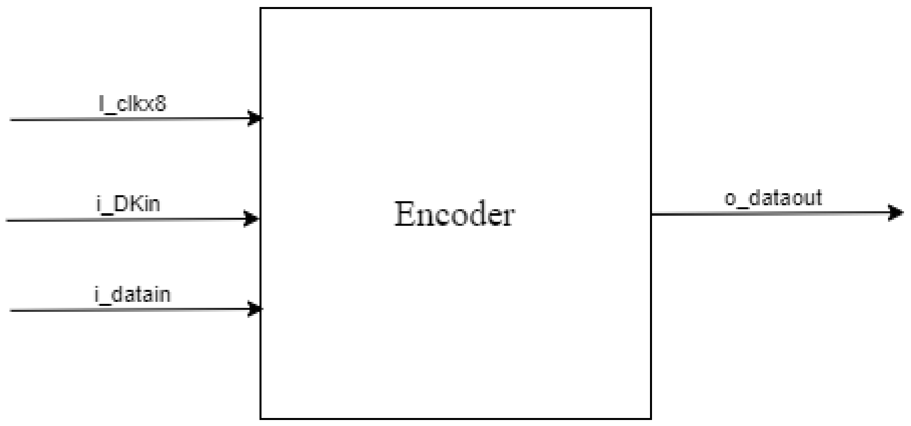


Figure 2: Block Diagram encoding.

2. Verification Approach

2.1. Testbench Architecture and Hierarchy

The testbench will follow standard UVM architecture. It will contain two agents, one for the interface between the Muxes IP interface and one for the interface between the PIPE and the Encoding modules of the Transmitter. Each agent will contain a Sequencer for managing the incoming sequence, a Driver for driving the interface, and monitor to sample the interface. Both monitors send their data to the scoreboard for error checking and to the subscriber for coverage collection. The schematic for the UVM environment is shown below.

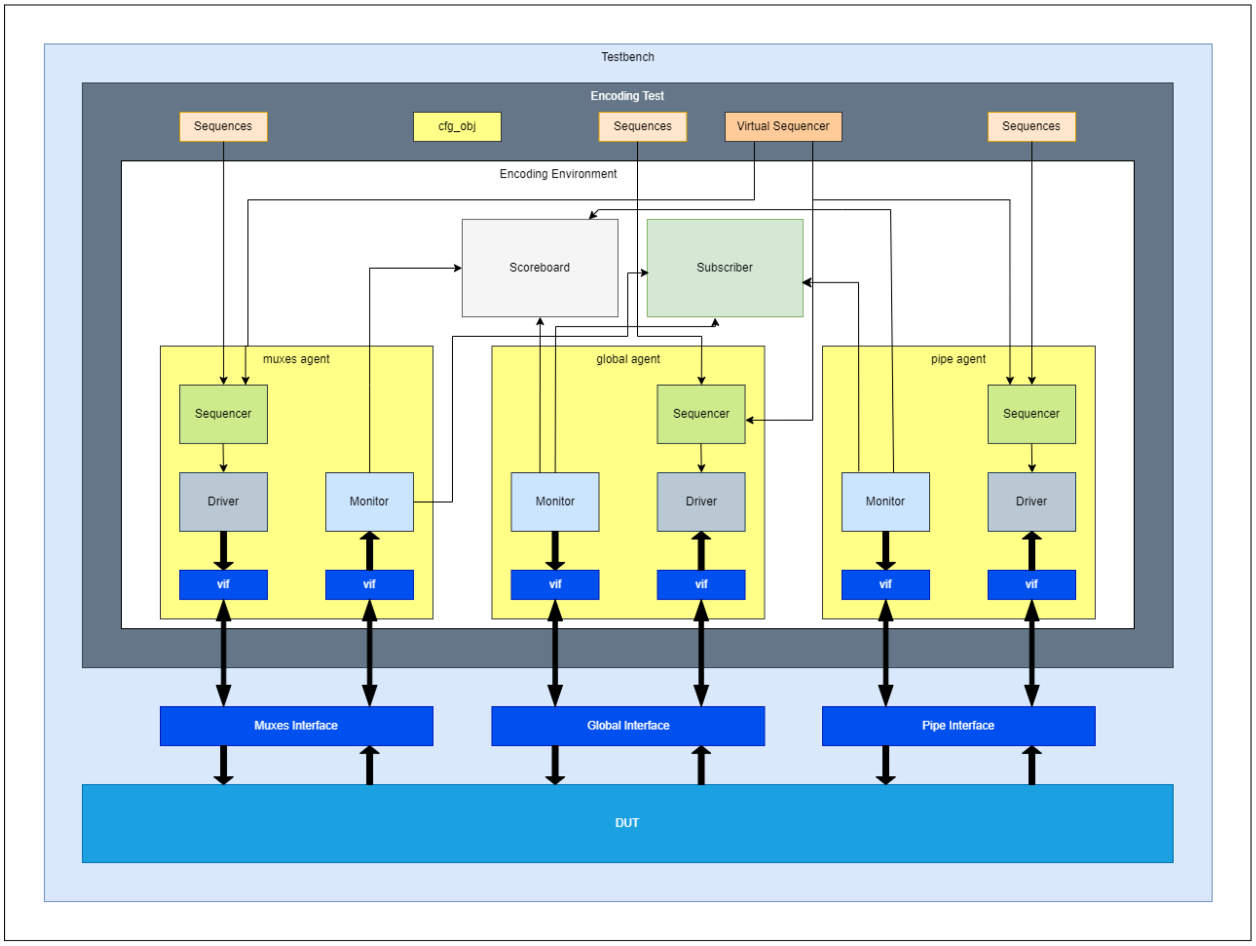


Figure 3: Architecture and Hierarchy encoding.

2.2. Exit Criteria

Since this is a simple module collection, to ensure that the module operation and functionalities are mostly covered, functional coverage of 100% and code coverage of 95% are required as exit criteria.

3. Test Items and Cases

The following are the designed test items:

- Reset sequence.
- Control data sequence.
- Normal date sequence.

Since this is a simple collection of modules, the UVM test will simply consist of sending a random series of these test items to test the behavior of the DUT.

4. Constraints

- The clockx8 signal is a secondary clock signal that is in-phase with the main clock but has 8 times the frequency.
- Only valid control signals are encoded by this module. But any data signal is allowed.

5. Checking Mechanism

The design will be compared against a golden reference model implemented in the scoreboard class using SystemVerilog. Any discrepancies will be recorded and reported.

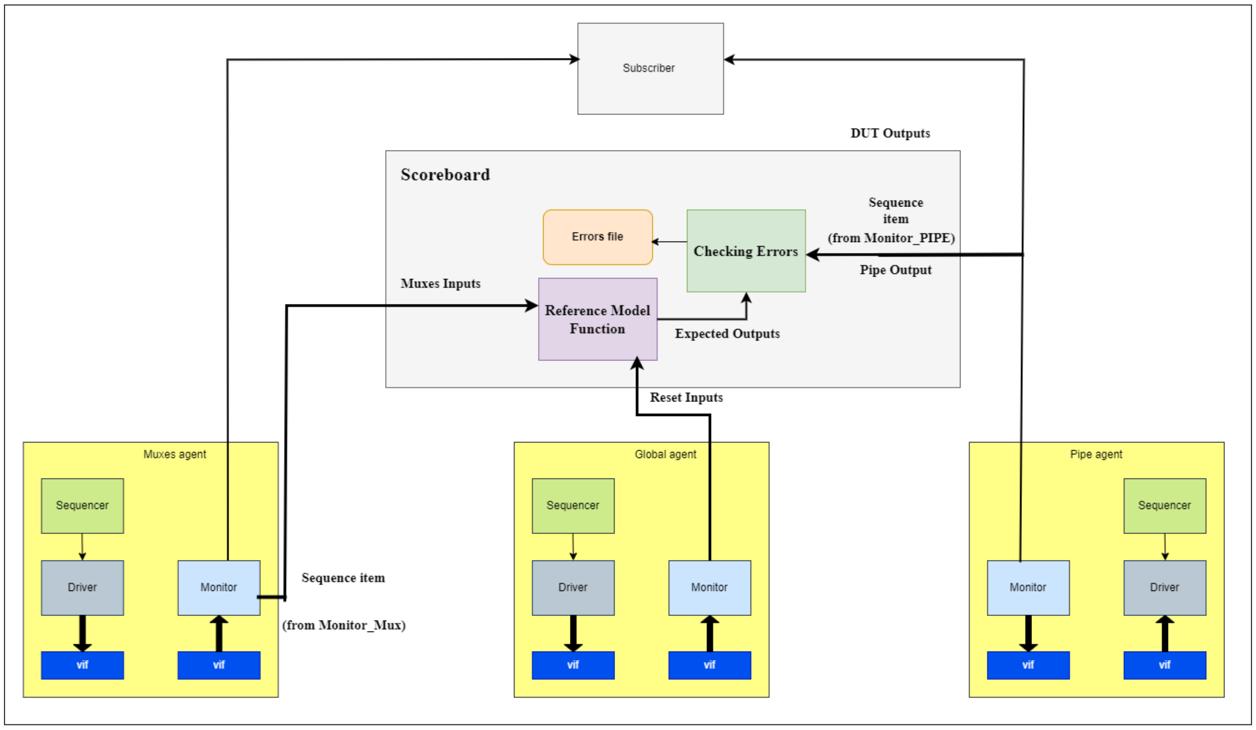


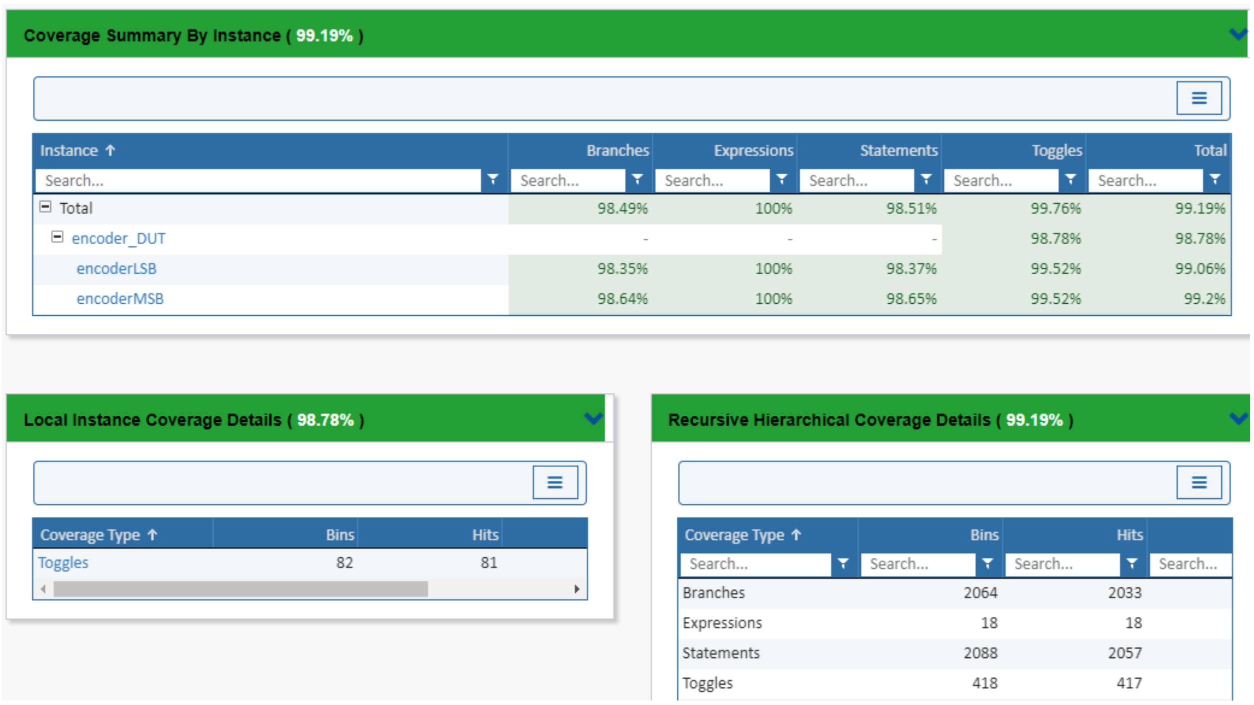
Figure 4: Checking mechanism encoding.

6. Coverage Collection

Functional coverage collection will be performed using the subscriber class. The table below shows the coverage bins needed for coverage collection.

Signal	Coverage Bins Description
i_reset_n_cp	1 bin for reset = 0, and 1 bin for reset = 1
i_Dkin_cp	1 bin for data = 10, and 1 bin for control = 01
i_datain	Auto bins for each of the possible values of i_datain
o_dataout_cp	Auto bins for ranges values of o_dataout

Code coverage will be performed automatically by the tool. The preliminary results are shown below.



Coverage Summary By Instance (99.19%)

Instance	Branches	Expressions	Statements	Toggles	Total
Total	98.49%	100%	98.51%	99.76%	99.19%
encoder_DUT	-	-	-	98.78%	98.78%
encoderLSB	98.35%	100%	98.37%	99.52%	99.06%
encoderMSB	98.64%	100%	98.65%	99.52%	99.2%

Local Instance Coverage Details (98.78%)

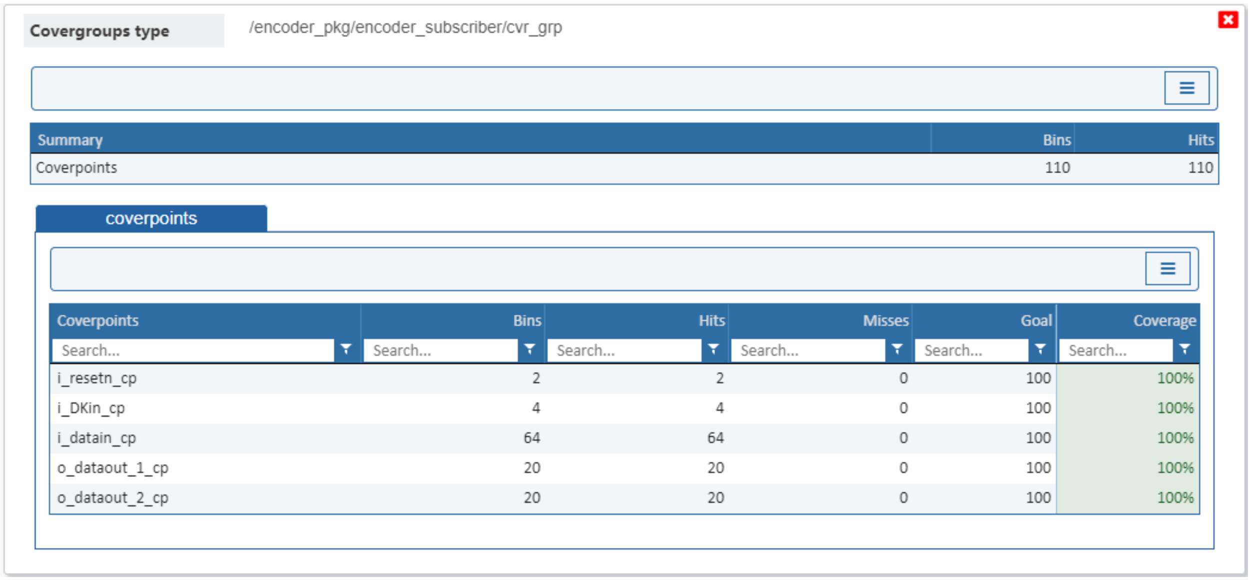
Coverage Type	Bins	Hits
Toggles	82	81

Recursive Hierarchical Coverage Details (99.19%)

Coverage Type	Bins	Hits
Branches	2064	2033
Expressions	18	18
Statements	2088	2057
Toggles	418	417

Figure 5: Code Coverage encoding.

Code Coverage is 99.19%, which is acceptable.


Covergroups type /encoder_pkg/encoder_subscriber/cvr_grp

Summary	Bins	Hits
Coverpoints	110	110

coverpoints

Coverpoints	Bins	Hits	Misses	Goal	Coverage
i_resetn_cp	2	2	0	100	100%
i_DKin_cp	4	4	0	100	100%
i_datain_cp	64	64	0	100	100%
o_dataout_1_cp	20	20	0	100	100%
o_dataout_2_cp	20	20	0	100	100%

Figure 6: Cover group encoding.

We can see that our test has achieved 100% coverage over the created cover groups.

7. Evaluation and Detected Bugs

This unit is: **GOOD TO GO**