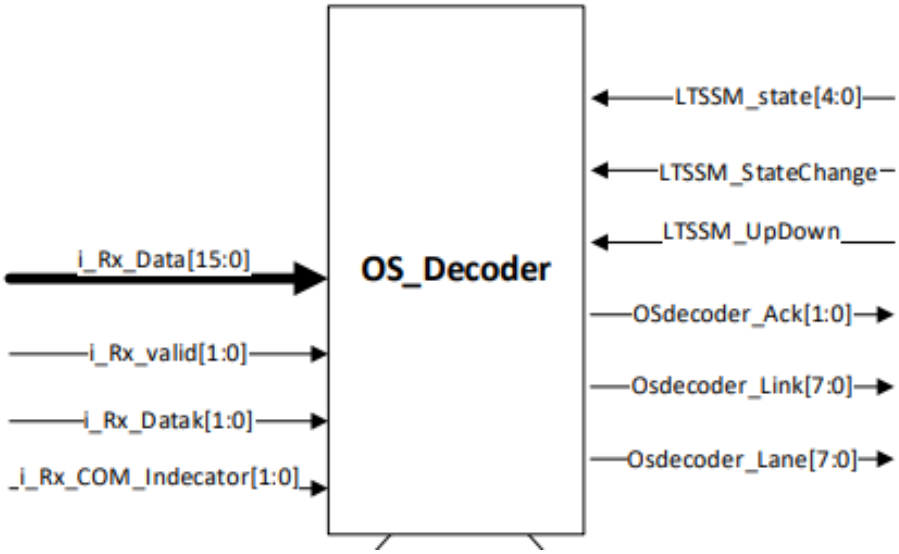
OS_Decoder IP Verification Plan

1. Introduction

1.1. Design Description

This IP receives data from the PCIe Rx and the state machine. Based on the data's content and the current state, it controls the state transitions during link training and initialization process.

1.2. Block I/O Signals



Inputs

Name	Width	Source	Description
clk	1 bit	Global	Clock signal
resetn	1 bit	Global	Synchronous low-level reset signal
i_Rx_Data	2 Bytes (16 bits)	PCIe Rx	Data from the receiver (sent from the other device on the link).

i_Rx_DataK	2 bits	PCIe Rx	Each bit corresponds to a byte in the Rx_Data signal. If the bit is 0, it corresponds to a data byte in the Rx_Data, and if it is 1, it corresponds to a control byte in the Rx_Data. Note: control bytes are COM, and bytes of Other OS (FTS, SKP, IDL).
i_Rx_valid	2 bits	PCIe Rx	Each bit indicates the validity of a byte in the Rx_Data signal.
i_Rx_COM_Indicator	2 bits	PCIe Rx	Indicates the beginning of a received ordered set in the Rx_Data signal. Note: 11 is an invalid case.
LTSSM_state	5 bits	State Machine	Encoding of the current state of the link.
LTSSM_StateChange	1 bit	State Machine	Indicates a change in the state of the link, so that the decoder should reset any counter of the received ordered sets to begin accounting only for those received in the new state.
LTSSM_UpDown	1 bit	State Machine	1 for Downstream and 0 for Upstream.

Outputs

Name	Width	Destination	Description
OSdecoder_Ack	2 bits	State Machine	Acknowledgement signal on the number of TS received to control the transition between the states. Note: for states requiring two conditions to occur, this signal is 01 when the first one occurs and 10 when the second one occurs. (e.g. Config_complete requires receiving one then eight TS2/Link#/Lane#.
OSdecoder_Lane	8 bits	State Machine	The lane number decoded from TS1 and TS2.
OSdecoder_Link	8 bits	State Machine	The link number decoded from TS1 and TS2.

2. Verification Approach

2.1. Verification Methodology

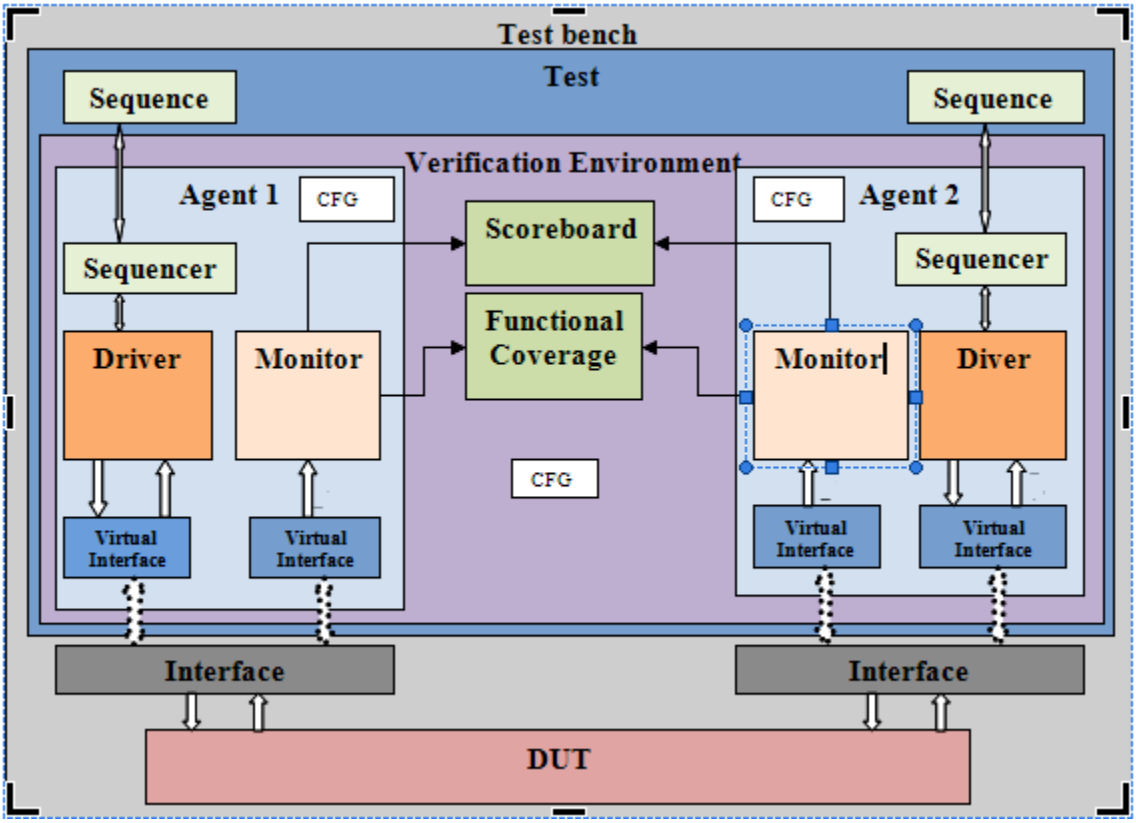
A coverage-driven constrained random functional verification methodology is employed. This approach allows for a more efficient use of resources by automating test generation, systematically exploring the design space, and uncovering unexpected corner cases and subtle bugs.

2.2. Verification Environment

A UVM-based verification environment is used. This offers standardization, modularity, reusability, scalability, and support for advanced verification techniques.

2.3. Testbench Architecture and Hierarchy

The testbench architecture will consist of 2 agents; one will be the normal agent that has its own drive, sequencer and monitor to verify the operation of this IP and the other agent is a global agent which will only work when we deactivate the first agent to test the overall functionality of the whole design as a one piece.



2.4. Tools to Be Used

Verification will be performed using Mentor QuestaSim as the primary simulator. Other tools such as Synopsys VCS may also be used to ensure compatibility with different simulators.

2.5. Exit Criteria

To ensure that the module operation and functionalities are mostly covered, functional coverage of 95% and code coverage of 100% are required as exit criteria.

3. Test Items

3.1 Symbols

In this subsection, I'll list here all symbols possible and then use these symbols to build test items and sequences as the i_Rx_Data consists of 2 symbols and each symbol is 8 bits.

Symbol Name	Encoding
PAD	8'hF7
TS1	8'h4A
TS2	8'h45
COM	8'hBC
SKP	8'h1C
FTS	8'h3C
IDL	8'h7C
EIE	8'hFC

3.2 Test items

The table below shows the test items that will be implemented and used in constructing the test cases.

Sequence Name	Value	Description
COM_before_config	i_Rx_Data = 16'hF7BC (Link#, COM) i_Rx_DataK = 2'b11 i_Rx_COM_Indicator = 2'b01 i_Rx_valid = 2'b11.	This is a combination for the COM symbol before the configuration state so the Link# should be PAD
COM_after_config	i_Rx_Data = 16'h01BC (Link#, COM) i_Rx_DataK = 2'b01 i_Rx_COM_Indicator = 2'b01 i_Rx_valid = 2'b11.	This is a combination for the COM symbol after the configuration state so the Link# should have a value

Symbols2_3_before_config	i_Rx_Data = 16'hxxF7 (N_FTS, lane#) i_Rx_DataK = 2'b01 i_Rx_COM_Indicator = 2'b00 i_Rx_valid = 2'b11.	This combination is for symbols 2 and 3 before the configuration state so the lane# should be PAD
Symbols2_3_after_config	i_Rx_Data = 16'hxx01 (N_FTS, lane#) i_Rx_DataK = 2'b00 i_Rx_COM_Indicator = 2'b00 i_Rx_valid = 2'b11.	This combination is for symbols 2 and 3 after the configuration state so the lane# should have a value
Symbols4_5	i_Rx_Data = 16'h0002 i_Rx_DataK = 2'b00 i_Rx_COM_Indicator = 2'b00 i_Rx_valid = 2'b11.	This combination is for symbols 4 and 5 which indicates Training Ctrl, and Rate ID. They have a fixed value which is 16'h0002
TS1	i_Rx_Data = 16'h4A4A i_Rx_DataK = 2'b00 i_Rx_COM_Indicator = 2'b00 i_Rx_valid = 2'b11.	This is the ID of TS1
TS2	i_Rx_Data = 16'h4545 i_Rx_DataK = 2'b00 i_Rx_COM_Indicator = 2'b00 i_Rx_valid = 2'b11.	This is the ID of TS1
Not_TS_nor_OS	i_Rx_Data = 16'hxxxx i_Rx_DataK = 2'bxx i_Rx_COM_Indicator = 2'b00 i_Rx_valid = 2'b11	This is when the input is not a TS, nor an OS and it is also not logical Idle
Logical_idle	i_Rx_Data = 16'h0000 i_Rx_DataK = 2'b00 i_Rx_COM_Indicator = 2'b00 i_Rx_valid = 2'b11.	This is the logical idle sequence

4. Test Cases

The table below shows the tests that will be applied to the design, the sequences used, and the description of each test. These test cases may contain several sequential inputs one after another.

Test Name	Sequences	Description
TS1/PAD/PAD	COM_before_config → Symbols2_3_before_config → Symbols4_5 → TS1 (x5)	This test case is the training sequence for the TS1 before configuration state
TS2/PAD/PAD	COM_before_config → Symbols2_3_before_config → Symbols4_5 → TS2 (x5)	This test case is the training sequence for the TS2 before configuration state
TS1/PAD/Link#	COM_after_config → Symbols2_3_before_config → Symbols4_5 → TS1 (x5)	This test case is the training sequence for the TS1 after configuration state where link# has a value while lane# is PAD
TS2/PAD/Link#	COM_after_config → Symbols2_3_before_config → Symbols4_5 → TS2 (x5)	This test case is the training sequence for the TS2 after configuration state where link# has a value while lane# is PAD
TS1/Lane#/Link#	COM_after_config → Symbols2_3_after_config → Symbols4_5 → TS1 (x5)	This test case is the training sequence for the TS1 after configuration state where lane# and link# must have values
TS2/Lane#/Link#	COM_after_config → Symbols2_3_after_config → Symbols4_5 → TS2 (x5)	This test case is the training sequence for the TS2 after configuration state where lane# and link# must have values

5. Sequence Layer

Complete sequence	Sequences	Description
Polling.Active_to_Polling.Config states	8x TS1/PAD/PAD LTSSM_state = 5'b00010	This sequence is the needed for the polling active state to go to the polling config state
Polling.Config_to_Configration states	8x TS2/PAD/PAD LTSSM_state = 5'b00011	This sequence is the needed for the polling config state to go to the polling configiration state
Linkwidth.start_to_Linkwidth.Accept States – downstream or upstream	2x TS1/PAD/Link# LTSSM_state = 5'b00100	This sequence is needed to go from the config.Linkwidth.start state to the config.Linkwidth.Accept state
Linkwidth.Accept_to_Lanenum.wait states - downstream or upstream	2x TS1/PAD/Link# LTSSM_state = 5'b00101	This sequence is needed to go from the config.Linkwidth.Accept state to the config.Lanenum.wait state
Lanenuum.wait_to_Lanenum.Accept States - downstream	2x TS1/Lane#/Link# LTSSM_state = 5'b00110 LTSSM_UpDown = 1'b1	This sequence is needed to go from the config.Lanenum.wait state to the config.Lanenum.wait state
Lanenum.Accept_to_Config.Complete States - downstream	2x TS1/Lane#/Link# LTSSM_state = 5'b00111 LTSSM_UpDown = 1'b1	This sequence is needed to go from the config.Lanenum.accept state to the config_complete state
Lanenuum.wait_to_Lanenum.Accept States - upstream	2x TS2/Lane#/Link# LTSSM_state = 5'b00110 LTSSM_UpDown = 1'b0	This sequence is needed to go from the config.Lanenum.wait state to the config.Lanenum.wait state
Lanenum.Accept_to_Config.Complete States - upstream	2x TS2/Lane#/Link# LTSSM_state = 5'b00111 LTSSM_UpDown = 1'b0	This sequence is needed to go from the config.Lanenum.accept state to the config_complete state

Config.complete_to_Config.idle States	1x TS2/Lane#/Link# then 8x TS2/Lane#/Link# LTSSM_state = 5'b01000	This sequence is needed to go from the config.complete state to the config idle state
Config.idle_to_L0	1x IDLE then 4x IDLE LTSSM_state = 5'b01001	This sequence is needed to go from the config_idle state to the L0 state

6. Constraints

- i_Rx_COM_Indicator cannot be 2'b11.
- LTSSM_state should not have different encodings than the following:

1. Detect_Quiet = 5'b00000
2. Detect_Active = 5'b00001
3. Polling_Active = 5'b00010
4. Polling_configuration = 5'b00011
5. Config_Linkwidth_start = 5'b00100
6. Config_Linkwidth_Accept = 5'b00101
7. Config_Lanenum_wait = 5'b00110
8. Config_Lanenum_Accept = 5'b00111
9. Config_complete = 5'b01000
10. Config_idle = 5'b01001
11. L0 = 5'b01010

7. 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.

8. Assertions

No assertions will be made in that IP.

9. 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

Code coverage will be performed automatically by the tool.

10. Traceability Matrix

11. Evaluation and Detected Bugs