AXI-I2S Transmitter

Verification Architecture Document

Author: **Muhammad Junaid**

10xEngineers

V1.0

Table of Contents

DUT Overview:

This project involves verifying the AXI I2S Transmitter Vivado IP. Verifying all the features outlined in the IP's specification documents. The IP should be generated in Vivado with default parameters. Developed a UVM based testbench for verifying this IP completely.
Implemented the RAL model for the registers. The other deliverables are follows:

- Git repository for project
- Testplan
- Coverage Model and Scores

Overview of I2S Transmitter:

Introduction:

The I2S (Inter-IC Sound) Transmitter is a digital audio interface that transmits audio data in the I2S format, commonly used for communication between digital audio devices such as processors, DACs, and audio codecs. The AXI-I2S Transmitter interface is designed to integrate with AXI4-Stream and AXI4-Lite protocols, providing a scalable and configurable solution for high-fidelity audio data transmission.

Features of I2S Transmitter

The I2S Transmitter includes the following features:

1. **Data Format and Protocol Support:**
    - Compliant with the I2S protocol.
    - Configurable data width: 16-bit or 24-bit audio samples.
    - Multi-channel audio support: Up to 8 audio channels (4 stereo pairs).
2. **Clock and Timing:**
    - Generates I2S-specific clocks (`SCLK`, `LRCLK`) derived from the master clock (`MCLK`).
    - Configurable `SCLK` divider to support various audio sampling rates.
3. **AXI4-Stream Interface:**
    - Receives 32-bit audio data with the following fields:
        - Bits [27:4]: Audio sample data (16 or 24 bits).
        - Bit [31]: Parity bit.
        - Bit [30]: Channel status.
        - Bit [28]: Validity bit.
        - Bits [3:0]: Preamble for block/channel identification.
4. **AXI4-Lite Interface:**
    - Used for configuration and control:
        - Enable/disable core operations.
        - Configure clock divider.
        - Enable/disable interrupts.

5. **Interrupt Support:**
    - Interrupts for the following events:
        - FIFO underflow.
        - AES channel status updates.
        - AES block synchronization errors.
        - AES block completion.
6. **Error Detection:**
    - Detection of invalid transactions (e.g., invalid TID, or preamble errors).
    - Block synchronization error handling.

Verification Features

To ensure the I2S transmitter functions as intended, the following key areas must be verified:

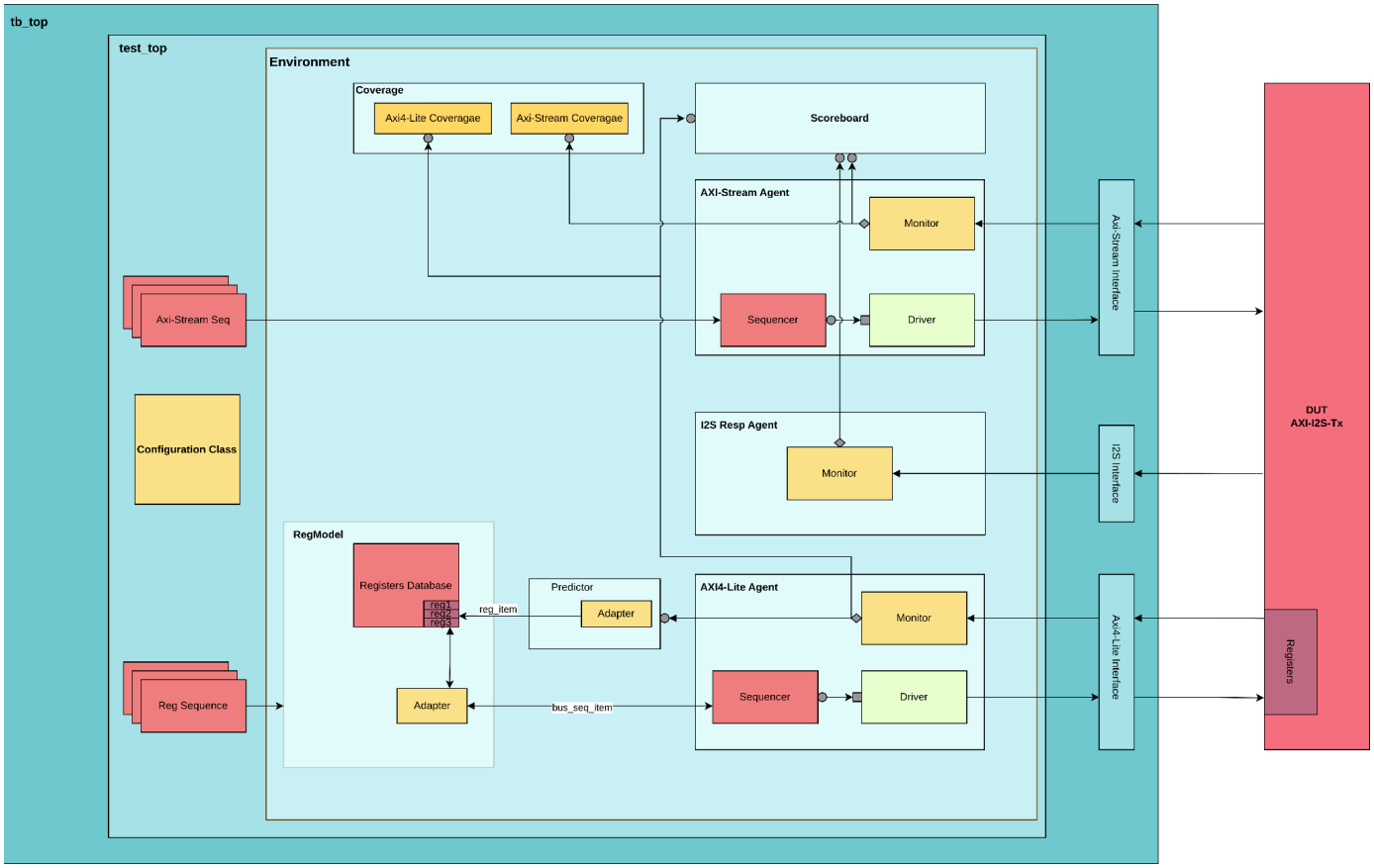
1. **AXI4-Stream Data Flow:**
    - Proper handshake using `tvalid` and `tready`.
    - Correct interpretation of preamble values (`4'b0001`, `4'b0010`, `4'b0011`) for block and channel identification.
    - Sequential and valid `TID` values for multi-channel audio.
    - Handling of validity bit (`TDATA[28]`).
2. **AXI4-Lite Configuration:**
    - Register read/write operations for core enable, clock divider configuration, interrupt settings.
    - Correct handling of invalid register addresses or unsupported configurations.
3. **Clock and Timing:**
    - Accurate generation of `SCLK` and `LRCLK` based on the configured divider.
    - Synchronization of data output with the generated clocks.
4. **Multi-Channel Audio:**
    - Correct routing and interleaving of up to 2 audio channels with default settings.
    - Validation of left/right channel alignment (`LRCLK` toggling).
5. **Error Scenarios:**
    - Detection and recovery from invalid preamble or TID values.

Test Bench Architecture

Overview:

The testbench architecture is designed to thoroughly validate the functionality and performance of the I2S Transmitter. The top-level module, `tb_top`, coordinates all testbench components and interfaces, ensuring seamless interaction between the DUT and verification environment.

A diagram illustrating the architectural flow and interactions between components is shown below:



Components

1. Interfaces:

- **AXI-Stream Interface:**
  - Facilitates the transmission of high-speed data streams from the testbench to the Design Under Test (DUT) using the AXI-Stream protocol.
  - Supports streaming data in a continuous flow with associated control signals like valid, ready, and data.
- **AXI4-Lite Interface:**
  - Provides a simplified, low-complexity connection between the testbench and the DUT for basic read and write operations on memory-mapped registers.
  - Supports a basic set of operations: read, write, and address management.
- **DUT Interface:**
  - Facilitates the connection to the Design Under Test (DUT) output signals, independent of AXI4-Lite and AXI-Stream, to monitor the DUT's output.

2. Test Class:

- Encapsulates the environment class and supports multiple test scenarios.
- Customized for specific test configurations based on verification objectives.
- All test cases are extended from a base test class.
- The base test efficiently uses UVM phases to reset, configure, and enable the core before executing extended tests.

3. Environment:

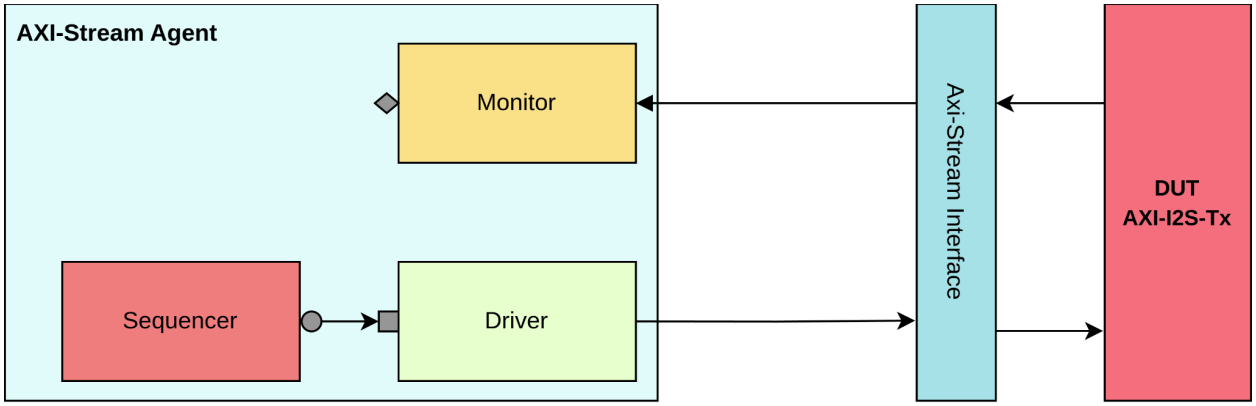
The environment class serves as the test environment, which provides the necessary context for verification. It integrates various agents like AXI Stream and AXI4-Lite, along with coverage and scoreboard components, to monitor and verify the interactions between the DUT and the testbench.

- **AXI-Stream Agent**

  The AXI Stream Agent is responsible for driving and monitoring the AXI Stream interface of the I2S transmitter. It includes the following components:

  - **Driver**: Sends transactions from the sequencer to the DUT through the AXI Stream interface.
  - **Monitor**: Monitors AXI Stream transactions and forwards interface data to the analysis port for coverage collection and scoreboard verification.
  - **Sequencer**: Provides stimulus to the driver by fetching sequence items from test sequences.

This agent ensures the proper handling of AXI Stream protocol transactions, including data, valid, and ready signals, while verifying compliance with the protocol.

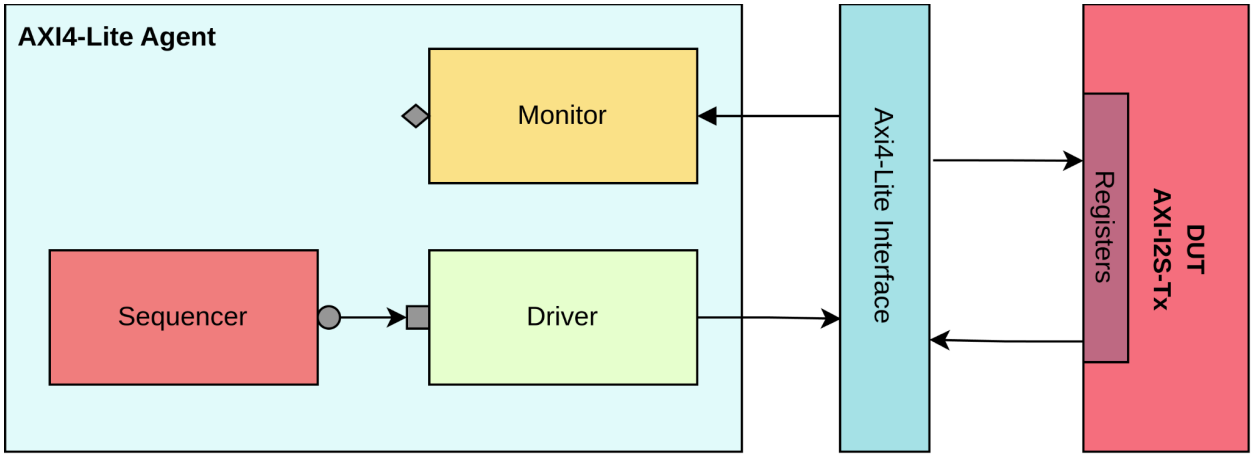


- **AXI4-Lite Agent**

  The AXI4-Lite Agent is responsible for managing register-level interactions with the I2S transmitter through the AXI4-Lite interface. It includes the following components:

  - **Driver**: Writes data to and reads data from the DUT registers by driving AXI4-Lite transactions.
  - **Monitor**: Passively observes and collects register transactions on the AXI4-Lite interface for coverage and validation against expected behavior.
  - **Sequencer**: Provides register-level stimulus to the driver by issuing sequence items, such as configuration and control commands.

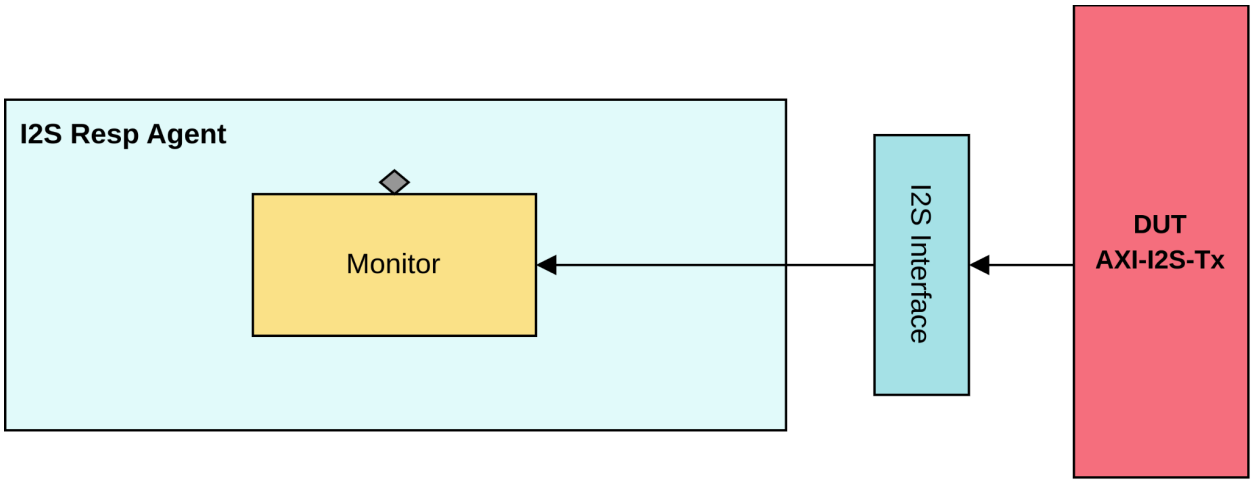
  This agent ensures the correct implementation of the AXI4-Lite protocol and enables configuration and monitoring of DUT behavior during verification.



- **DUT Response Agent**

  The DUT Response Agent is responsible for monitoring and verifying the responses generated by the DUT (I2S transmitter). It operates in a passive mode and consists of the following component:

  - **Monitor**: Observes the DUT interface signals, collects response data, and transfers it to the scoreboard for comparison against expected results.

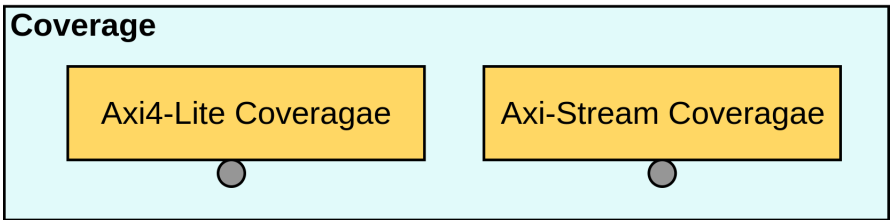


- **Coverage**

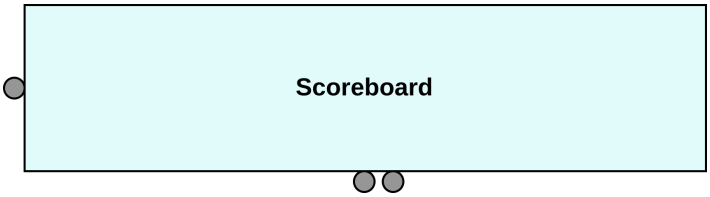
  The coverage component ensures that all functional and protocol aspects of the I2S transmitter are thoroughly validated. It tracks whether all required scenarios have been exercised during verification. The coverage is divided into the following categories:

  - **AXI4-Lite Coverage:**
    - Ensures all registers are accessed, including valid and invalid read/write operations.

        - Tracks coverage for control bits, configuration parameters, and register interactions.
        - Monitors protocol compliance for AXI4-Lite signals (e.g., `AWVALID`, `ARREADY`, `WVALID`, `RREADY`).
    - **AXI-Stream Coverage**
        - Tracks the data flow, including valid preambles, channel data, and audio sample transactions.
        - Ensures sequential and valid `TID` values for multi-channel audio data.
        - Monitors handshake signal combinations (`TVALID`, `TREADY`) and their alignment with the protocol



- **Scoreboard**
    - The scoreboard is a critical component in the testbench responsible for comparing the expected and actual outputs of the DUT to validate its behavior. It acts as a central repository for tracking and verifying the correctness of data transactions



4. Sequence

The sequence defines the stimulus provided to the DUT, controlling the data and transactions on both the AXI4-Stream and AXI4-Lite interfaces. It is responsible for generating directed, random, and constrained random scenarios to thoroughly test the I2S transmitter.

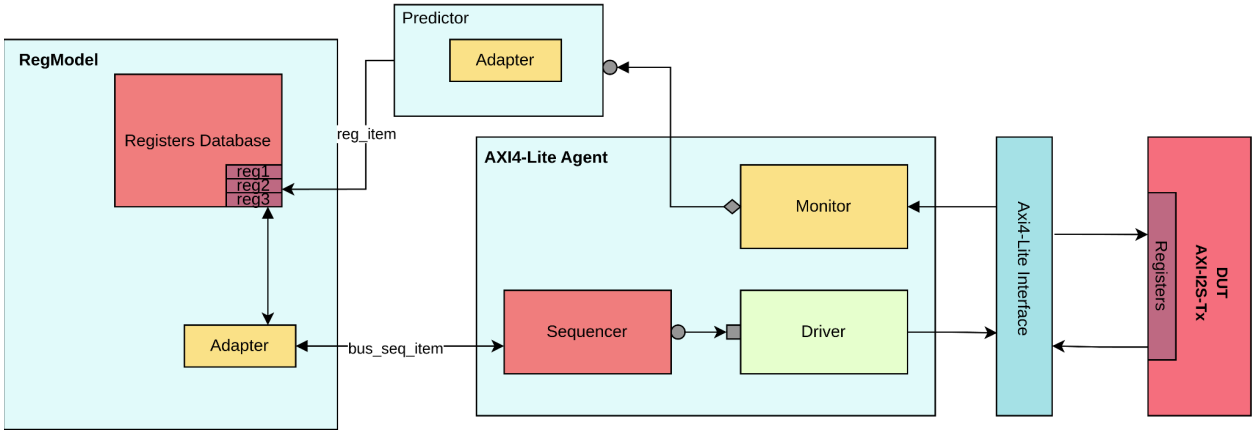
- **AXI-Stream Sequence**
    - Drives audio data into the DUT via the AXI-Stream interface.
    - Stimulates the DUT with various patterns, including:
        - Valid preamble values (`4'b0010`, `4'b0011`).
        - Audio sample data for different data widths (16-bit, 24-bit).
        - Validity, parity, and channel status bits.

        - Tests multi-channel configurations (2 channels) with correct interleaving of left and right channel data.
- **Register Sequence**
    - Performs configuration and control operations via the AXI4-Lite interface.
    - Stimulates:
        - Core enable/disable operations.
        - Clock divider configuration for generating SCLK.
        - Interrupt enable/disable and status checking.

5. RegModel

The **Reg Model** is a representation of the DUT's register map, used for abstracting and managing register interactions during verification. It provides a high-level mechanism to read, write, and verify the values of the DUT's registers.

- **Registers Database**
    - Defines the structure and attributes of all registers in the DUT, including address, access type (read/write), and default values.
    - Includes all key registers of the I2S transmitter, such as:
        - Core Version Register
        - Core Configuration Register
        - Control Register
        - Validity Register, etc
- **Adapter**
    - Translates high-level register transactions (read/write) into AXI4-Lite protocol signals for interaction with the DUT.
    - Ensures that all AXI4-Lite register transactions comply with the protocol.
- **Predictor**
    - Mirrors the expected behavior of the DUT's registers.
    - Predicts the expected register values based on stimuli applied through the AXI4-Lite agent, ensuring consistency and correctness. Translate Bus transactions to register level transactions using adapter



6. **Configuration Class**
    - The configuration class manages settings for the test bench, including enabling and controlling various test bench features. It ensures the environment is set up correctly for simulation and testing.

Test Bench Environment Directory Structure:

The Following is test bench environment directory structure

```
├──Coverage
│      └── xsim.covdb
├── coverage_merged
│      └── xsim.covdb
├── coverage_report
│      └──*.html
├── doc
│      └──*.pdf
├── dut
│      └── I2S_IP
├── logs
│      └── test_logs
├── resources
│      └──*.png
└── tb
    ├── coverage
    ├── env
    ├── I2S_RSP_AGENT
    ├── include
    ├── scoreboard
    ├── test_top
    ├── top
    ├── UVC_AXI4_LITE
    │   └── UVM_RAL
    └── UVC_AXI_STREAM
```

Test Workflow

High-Level Overview:

The test workflow describes the systematic process to validate the I2S transmitter against its specifications. Each test follows these steps:

1. **Test Initialization:**
    - The testbench components and environment are set up.
    - DUT is configured using the AXI4-Lite interface, and reset sequences are applied.
2. **Stimulus Generation:**
    - AXI-Stream sequences generate audio data transactions.
    - AXI4-Lite sequences configure registers and verify control operations.
3. **Response Monitoring:**
    - The DUT's outputs (`SCLK`, `LRCLK`, `SDATA`) are monitored.
    - Transactions are validated for protocol adherence and correctness using monitors and the scoreboard.
4. **Coverage Collection:**
    - Functional and protocol coverage is tracked throughout the test.
5. **Results Analysis:**
    - Logs and coverage reports are generated, and any mismatches or failures are analyzed.

Detailed Overview

Step 1: Testbench Initialization

- The testbench is instantiated from the base test class (`i2s_tx_base_test`).
- The top-level testbench (`i2s_tx_tb_top`) initializes the DUT and its interfaces.
- Configuration parameters are passed to the UVM configuration database for use by environment components.

Step 2: Build Phase

- Instantiate Agents:
    - Create AXI4-Lite, AXI-Stream, and I2S agents to handle DUT interactions.
    - Verify successful creation and log any issues.
- Create Register Model:
    - Instantiate the register block (`reg_block`) to abstract the DUT's registers.
    - Build and print the register model for debugging purposes.
- Setup Predictor and Adapter:

    - Create a predictor to simulate register behavior and compare expected vs. actual values.
    - Instantiate an adapter to bridge the register model with the AXI4-Lite protocol.
- Instantiate Scoreboard and Coverage Components:
    - Create the scoreboard for end-to-end validation of DUT responses.
    - Instantiate AXI4-Lite and AXI-Stream coverage classes if coverage collection is enabled.
- Retrieve Configuration:
    - Fetch the configuration class (`i2s_tx_config`) from the UVM configuration database.

Step 3: Connect Phase

- Connect Analysis Ports:
    - Link monitor analysis ports of the agents to the predictor, scoreboard, and coverage components.
- Configure Register Block:
    - Set the sequencer, adapter, and base address for the register block.
    - Ensure the predictor is properly linked to the register map for accurate validation.

Step 4: Test Execution (Run Phase)

- Reset the DUT:
    - Execute the reset sequence to initialize the DUT to a known state.
- Configure the DUT:
    - Run the configuration sequence to set up registers such as enabling the core, setting clock dividers, and enabling interrupts.
- Stimulus Generation:
    - The AXI-Stream agent drives data transactions to the DUT, including valid audio frames and preamble.
    - The AXI4-Lite agent performs register-level read and write transactions.
- Monitor DUT Responses:
    - Capture responses from the DUT using monitors on AXI4-Lite, AXI-Stream, and I2S interfaces.
    - Feed captured data to the scoreboard for validation and coverage collection.

Step 5: Validation and Analysis

- Scoreboard Validation:
  - Compare DUT outputs (e.g., I2S signals like `SCLK`, `LRCLK`, `SDATA`) with expected results.
  - Flag any mismatches or errors in the logs.
- Coverage Collection:
  - Track functional and protocol coverage for AXI4-Lite and AXI-Stream transactions.
  - Ensure all features, scenarios, and corner cases are exercised.

Step 6: Completion (Report Phase)

- Generate a final report summarizing:
  - Test pass/fail status based on UVM error and fatal counts.
  - Coverage reports highlighting tested and untested areas.
  - Detailed logs of transactions, errors, and assertions.

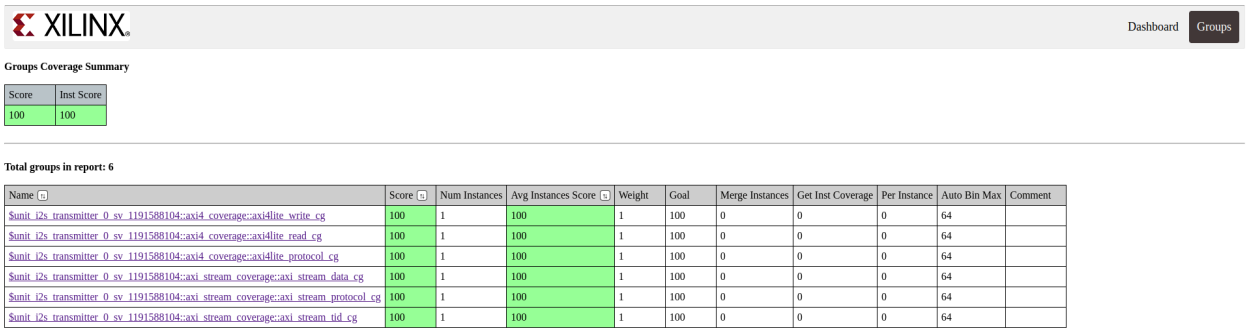
Test Plan

To view in pdf format visit this link

AXI_I2S_TX_Test Plan

Coverage

For detailed coverage report in html format visit this link



XILINX

Dashboard Groups

Groups Coverage Summary

Score	Inst Score
100	100

Total groups in report: 6

Name	Score	Num Instances	Avg Instances Score	Weight	Goal	Merge Instances	Get Inst Coverage	Per Instance	Auto Bin Max	Comment
$unit_i2s_transmitter_0_sv_1191588104::axi4_coverage::axi4lite_write_cg	100	1	100	1	100	0	0	0	64	
$unit_i2s_transmitter_0_sv_1191588104::axi4_coverage::axi4lite_read_cg	100	1	100	1	100	0	0	0	64	
$unit_i2s_transmitter_0_sv_1191588104::axi4_coverage::axi4lite_protocol_cg	100	1	100	1	100	0	0	0	64	
$unit_i2s_transmitter_0_sv_1191588104::axi_stream_coverage::axi_stream_data_cg	100	1	100	1	100	0	0	0	64	
$unit_i2s_transmitter_0_sv_1191588104::axi_stream_coverage::axi_stream_protocol_cg	100	1	100	1	100	0	0	0	64	
$unit_i2s_transmitter_0_sv_1191588104::axi_stream_coverage::axi_stream_tid_cg	100	1	100	1	100	0	0	0	64	

Resources

GitHub Repository

AMBA AXI-Stream Protocol Specification

I2S-Spec

AXI-Spec