

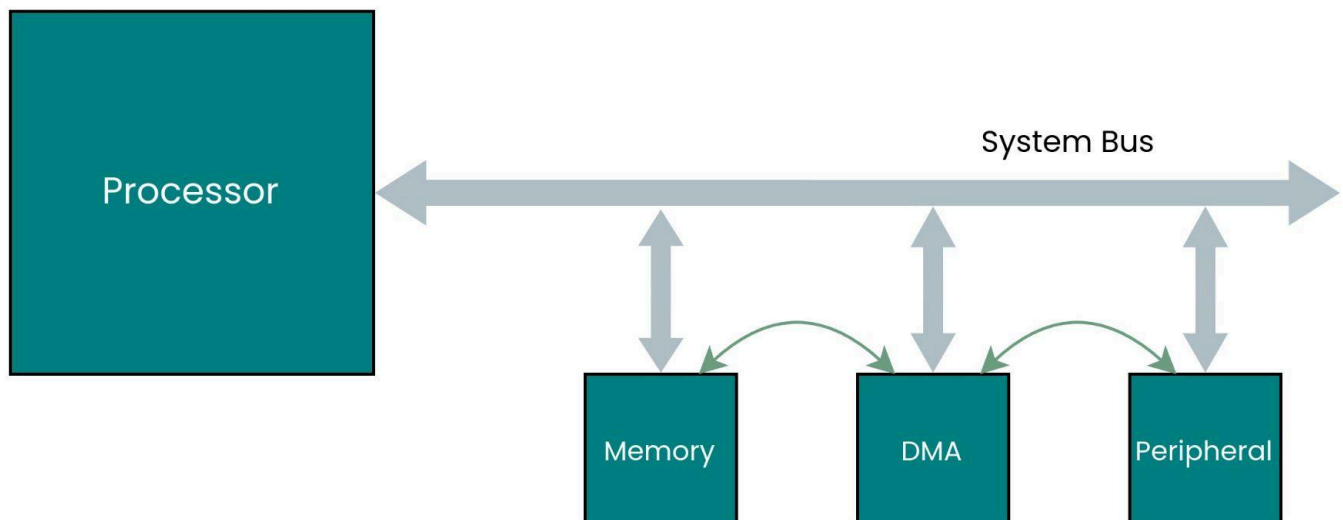
# AXI DMA Verification

Status Completed ▾

Timing Dec 12, 2024 to Jan 31, 2025

Owners Noman Rafiq

Tech Lead: Hassan Ashraf



# Table of Contents

<b>Table of Contents.....</b>	<b>1</b>
<b>1. Overview:.....</b>	<b>2</b>
<b>2. Features to Verify:.....</b>	<b>2</b>
<b>3. Testbench Architecture:.....</b>	<b>3</b>
3.1 Overview.....	3
3.2 Components.....	4
1. Interfaces.....	4
2. Block RAM (BRAM).....	4
3. Test Class.....	5
4. Environment Class.....	5
3.3 Detailed Components.....	6
3.1.1. Register_UVC.....	6
3.1.2. RAL Model.....	7
3.1.3. Scoreboard.....	7
3.1.4. Stream_UVC.....	8
<b>4. Folder Structure Diagram:.....</b>	<b>9</b>
<b>5. Test Workflow.....</b>	<b>10</b>
5.1. High-Level Overview.....	10
5.2. Detailed Workflow.....	10
5.2.1. Test Initialization.....	10
5.2.2. Run Phase.....	11
• <b>DUT Configuration:</b> .....	<b>11</b>
• Sequence Execution:.....	11
• Monitoring:.....	11
• Scoreboarding:.....	11
• Coverage Model:.....	12
• Termination:.....	12
<b>6. Testplan.....</b>	<b>12</b>
<b>7. Coverage Scores.....</b>	<b>13</b>
<b>8. Verification Challenges.....</b>	<b>13</b>
<b>9. Timeline.....</b>	<b>13</b>
<b>10. References:.....</b>	<b>14</b>

## AXI DMA Verification Verification Architecture Document

### UVM Verification Microarchitecture AXI DMA

---

#### 1. Overview:

This project focuses on the verification of the AXI DMA IP designed in Vivado, with a specific emphasis on the **Direct Register Mode** functionality. The **Scatter/Gather Engine** is intentionally disabled to streamline the verification scope and center efforts on validating the core functionality of the DMA in this configuration.

The primary objective of the project is to ensure that the AXI DMA operates correctly and adheres to the defined specifications. This involves comprehensive testing to identify and resolve potential design issues, thereby achieving functional correctness, protocol compliance, and verification closure through coverage metrics.

##### High-level goals include:

- 1.1. **Functionality Verification:** Ensuring the DMA performs data movement between memory and AXI-Stream interfaces as expected.
- 1.2. **Protocol Compliance:** Validating adherence to the AMBA AXI and AXI-Stream specifications.

#### 2. Features to Verify:

The key features to be verified include:

##### 2.1. Read/Write Operations for AXI-Lite Interface

Verifying the AXI DMA's ability to correctly configure and respond to control and status register read/write operations using the AXI-Lite interface.

##### 2.2. Data Transfer Using AXI-Stream Interface

Ensuring seamless data movement between AXI-Stream interfaces and system memory, including packetized streaming modes.

### 2.3. **Handling Packet Boundaries**

Validating proper recognition, processing, and termination of packets during data transfer, ensuring alignment with stream-based data protocols.

### 2.4. **Interrupt Generation and Handling**

Testing the DMA's ability to generate interrupts for events such as transfer completion, errors, and other status updates. Verifying the correct handling and acknowledgment of these interrupts in the design.

### 2.5. **Error Scenarios**

Evaluating the DMA's behavior under error conditions such as buffer overflows.

### 2.6. **Reset Feature**

Verifying proper reset behavior, ensuring all registers return to their default state.

### 2.7. **MM2S/S2MM Independent Operation**

Ensuring that MM2S (Memory-Mapped to Stream) and S2MM (Stream to Memory-Mapped) transfers function independently without interference.

### 2.8. **4 KB Boundary Protection**

Validating AXI4 address boundary rules, ensuring transfers do not cross the 4 KB boundary, as required by the AXI specification.

### 2.9. **Automatic Data Re-Alignment**

Confirming that the DMA properly handles unaligned data accesses, allowing reads/writes to begin at any byte offset while preserving data integrity.

### 2.10. **Halted and Idle State**

Ensuring that the DMA transitions to the Halted or Idle state correctly after transfers stop or errors occur. Validating status flags that indicate DMA state transitions.

### 2.11. Run/Stop Control

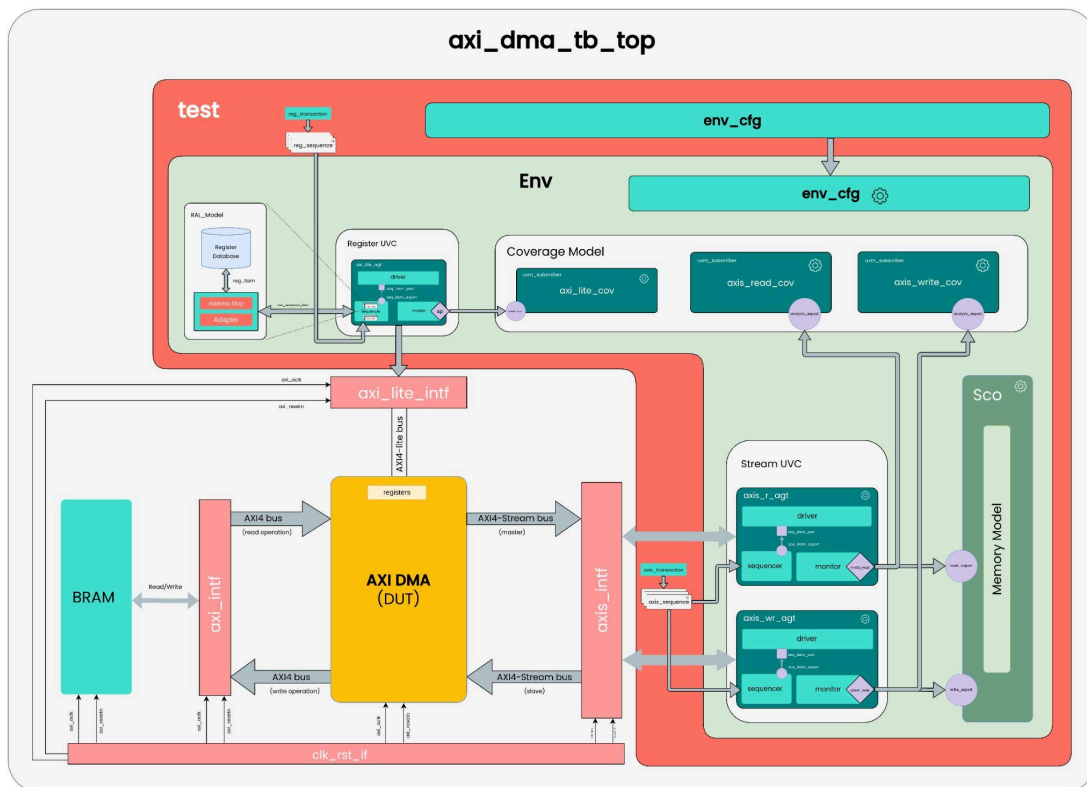
Verifying that the Run/Stop control bits in the DMACR (DMA Control Register) correctly start and stop transfers without causing data corruption.

## 3. Testbench Architecture:

### 3.1 Overview

The testbench architecture is designed to validate the functionality and performance of the AXI DMA IP comprehensively. The top-level module, **axi\_dma\_tb\_top**, orchestrates all testbench components and interfaces, ensuring a streamlined interaction between the DUT and verification components.

An image illustrating the architectural flow and component interaction is attached below:





## 3.2 Components

### 1. Interfaces

- **clk\_rst\_if** (Clock and Reset Interface): Used for clock and reset generation.
- **axi\_lite\_intf** (AXI4-Lite): Used for control and status register configuration.
- **axi\_intf** (AXI4): Handles memory-mapped data transactions for the DMA.
- **axis\_intf** (AXI-Stream): Facilitates data transfer between the DUT and external systems using AXI-Stream protocol.

### 2. Block RAM (BRAM)

- Serves as system memory for memory-mapped read/write operations.
- Acts as the primary data repository for testing the DMA's memory interfaces.

### 3. Test Class

- Encapsulates the environment class and supports multiple test scenarios.
- Customized for specific test configurations based on verification objectives.



- 
- The diagram illustrates the Env framework architecture. It begins with a 'MIPS UVM?' block, which connects to a 'RAL\_Model' block. The 'RAL\_Model' block contains a 'Register Database' and an 'Abstract Any' component. The 'RAL\_Model' interacts with a 'Register UVC' block, which in turn connects to a 'Coverage Model' block. The 'Coverage Model' contains several sub-models, including 'uvm\_subscriber' and 'uvm\_monitor'. These models interact with a 'Stream UVC' block, which contains 'axis\_r\_agt' and 'axis\_wr\_agt' components. The 'Stream UVC' block also interacts with a 'Memory Model' block, which is part of the 'Sco' (System-on-Chip) environment. The entire process is managed by an 'env\_cfg' configuration block.

### 3.3 Detailed Components

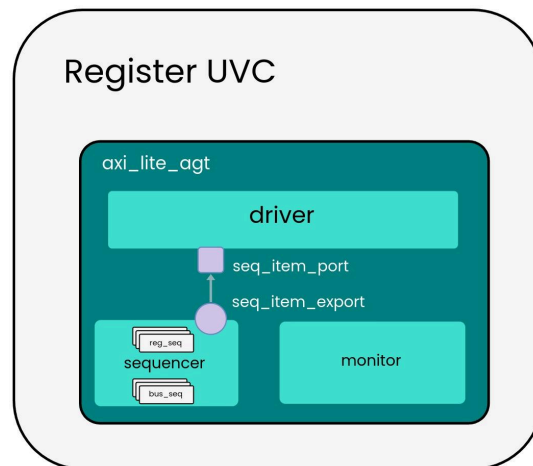
#### 3.1.1. Register\_UVC

Encapsulates the **axi\_lite\_agt** responsible for configuring the DMA core. The agent consists of:

**Driver:** Drives transactions from the sequencer onto the AXI-Lite interface to read/write registers in the DUT.

**Sequencer:** Fetches transactions from the sequence library or RAL\_Model.

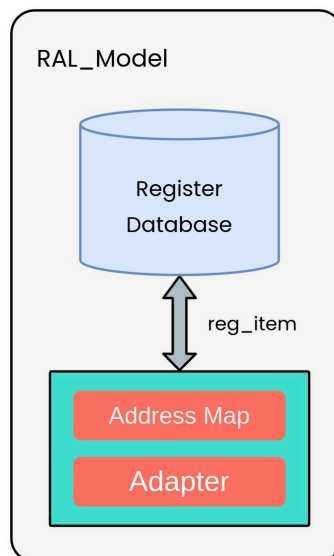
**Monitor:** Observes AXI-Lite transactions for validation and coverage collection.



#### 3.1.2. RAL Model

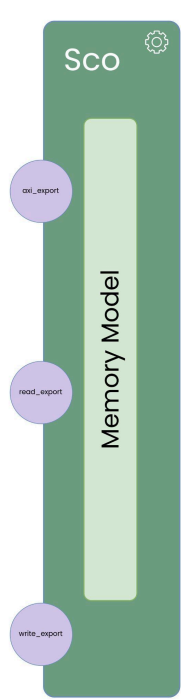
Mimics the memory-mapped registers of the DMA core, providing a high-level abstraction for register access.





### 3.1.3. Scoreboard

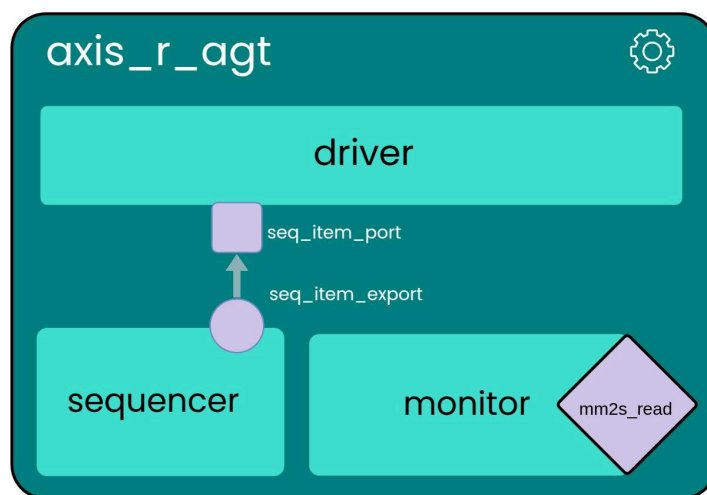
- Maintains a local memory model for read/write operations.
- Compares data read from local memory to actual data on the AXI-Stream read channel.
- Mirrors transactions written to the BRAM in its local memory for validation.
- Ensures data integrity by comparing expected and actual results, determining test pass/fail status.



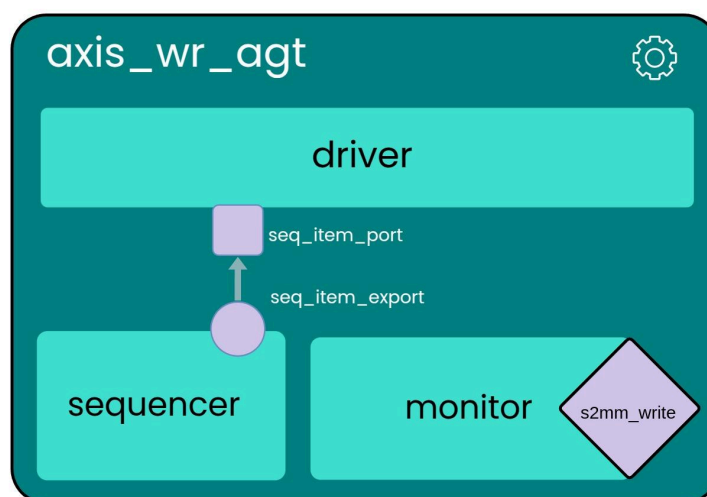
### 3.1.4. Stream\_UVC

Contains two agents to verify AXI-Stream channels:

**axis\_read\_agent:** Handles transactions for the MM2S (Memory-Mapped to Stream) Read Channel.

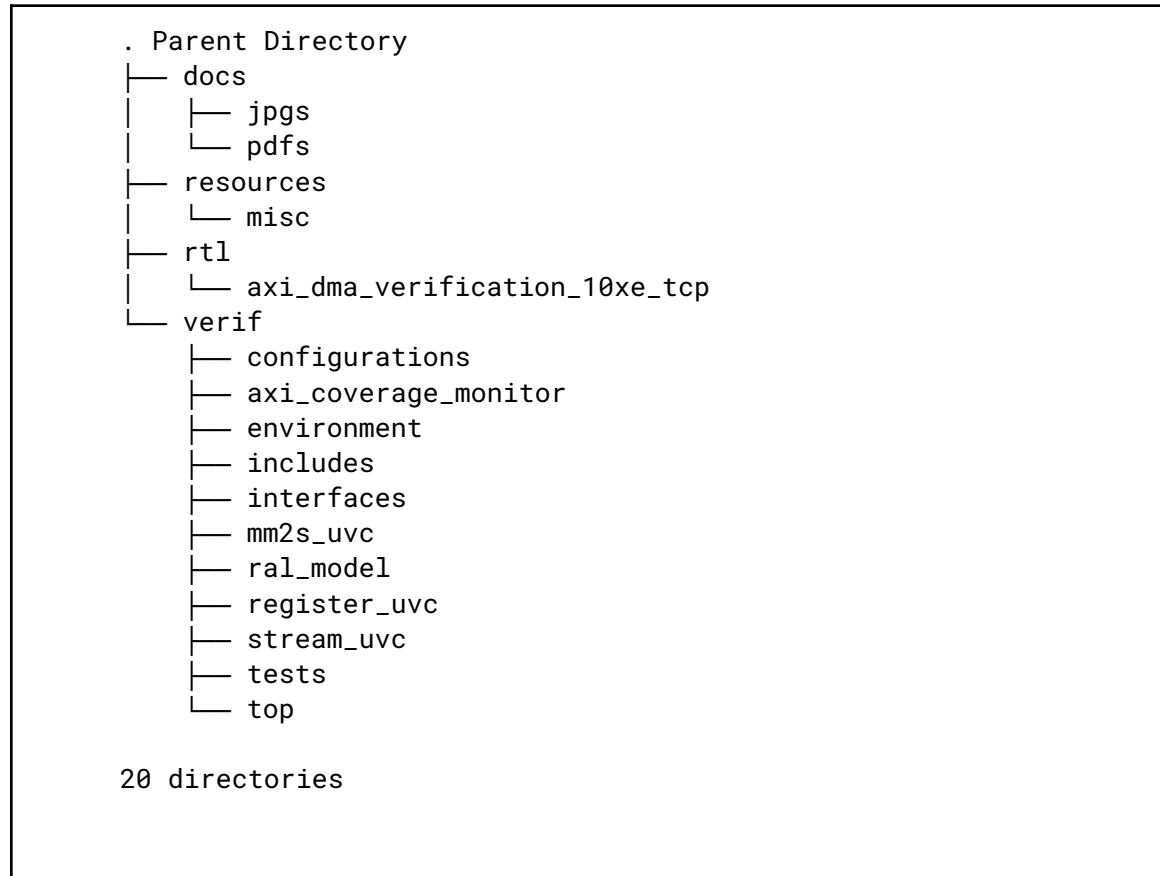


**axis\_write\_agent:** Manages transactions for the S2MM (Stream to Memory-Mapped) Write Channel.



#### 4. Folder Structure Diagram:

The verification environment has the following folder structure:



#### 5. Test Workflow

##### 5.1. High-Level Overview

The test workflow begins in the **tb\_top** module by invoking the **run\_test("test")** method. This initiates the execution of the **base\_test** class, which orchestrates key setup and configuration tasks required for the simulation.

The workflow ensures the proper configuration of interfaces, sequence execution, and result validation. Interfaces such as AXI, AXI-Lite, and AXI-Stream are configured using the **uvm\_config\_db**, enabling communication between components like drivers, monitors, and the DUT.

## 5.2. Detailed Workflow

### 5.2.1. Test Initialization

- The test begins with the invocation of **run\_test("base\_test")** in the **tb\_top** module. This triggers the execution of the **build\_phase** across all hierarchical components.
- During the **build\_phase** of the test, all components are created and connected through their **connect\_phase** (bottom-up).
- A dynamic configuration object (**env\_cfg**) is modified to test specific scenarios.

For instance: To enable **MM2S\_Read** operation, the **SRC\_ADDR** and the number of **bytes** to read must be specified. The scoreboard's write operation is disabled to avoid entering an infinite loop.

The number of transactions (**num\_trans**) is calculated as **num\_trans = env\_cfg.calculate\_txns()** within the **build\_phase**. This calculation ensures that the testbench operates without any infinite loops.

The testbench configuration is derived from this **env\_cfg** file, ensuring all components are tailored for the scenario being verified.

### 5.2.2. Run Phase

- **DUT Configuration:**
  1. A simulation objection is raised at the start of the **run\_phase**.
  2. DUT control and status registers are configured to enable desired operations such as **MM2S** (Memory-Mapped to Stream) Read or **S2MM** (Stream to Memory-Mapped) Write.
- **Sequence Execution:**
  1. Sequences such as **axis\_read** and **axis\_write** are started on their respective sequencers.
  2. These sequences drive transactions onto the AXI-Stream interfaces for data transfer operations.
- **Monitoring:**
  1. Monitors in each agent observe activity on their respective interfaces.
  2. Valid transactions are written to the analysis port by invoking the **write()** method, making them available


for further analysis by other components like the scoreboard.

- **Scoreboarding:**
  1. The scoreboard captures transactions broadcasted via TLM implementation ports.
  2. It performs a comparison between captured transactions and expected results, identifying any mismatches.
  3. Errors, if any, are reported, and the test pass/fail status is determined during the **report\_phase**.
- **Coverage Model:**
  1. The coverage model subscribes to transactions via TLM analysis ports by implementing the write() method.
  2. These transactions drive the sampling of functional coverage groups defined in the coverage model.
  3. Coverage groups capture key protocol behaviors and interface activities, including:
    4. AXI-Stream protocol events: Handshake patterns, burst lengths, and packet boundaries.
    5. Control signals: Activity of tvalid, tready, and interrupt signals (mm2s\_introut, s2mm\_introut).
    6. This systematic sampling ensures that all critical scenarios and edge cases are exercised during the verification process.
- **Termination:**
  1. Once all sequence items are driven and monitored on the interfaces, the test workflow transitions to termination.
  2. The simulation objection is dropped, signaling the end of the test.

## 6. Testplan

Testplan_AXI_DMA_VERIF									
AXI DMA - Test Plan									
Test ID	Feature	Test Name	Description	Input Stimulus	Expected	Checking Procedure	Status	Comments	
1	Reset	reset_test	When reset is asserted, the DMA Core is resetted and no read or write operation can happen unless configured.	Apply clock; axi_resetn = 0; Execute Read_status Sequence	DMA core should read default values for S2MMSTR and DMASTR Registers. The read value should be 'h1' indicating both channels halted.	Checkers in register Sequences can validate the reset state of the DMA Core. Can also be confirm via waveform.	Pass		
Independent Operation									
2	MM2S Enable	mm2s_enable_test	Enables the Memory Mapped to Stream Read Operation. The dut will start reading from the provided base address and streams the data on the AXI-Stream interface.	Configure the DMA's internal registers as following: 1. MM2S_DMCR = 32'h11001 2. MM2S_SA = 32'h0 3. MM2S_LENGTH = 32'h80	The DMA should assert <b>arvalid</b> alongwith valid <b>SRC_ADDR</b> to make a read request to AXI Slave Memory.	The signal activity can be monitored from waveform.	Pass		
3	S2MM Enable/Write Test	s2mm_enable_test/write_test	Enables the Stream to Memory Mapped Write Operation. The dut will take stream as input and start writing on the provided base address via AXI-4 interface.	Configure the DMA's internal registers as following: 1. S2MM_DMCR = 32'h11001 2. S2MM_DA = 32'h0 3. S2MM_LENGTH = 32'h80 4. Start Axis_write Sequence	The DMA should be able to accept the write transactions through <b>valid &amp; ready</b> handshake according to AXI-Stream Protocol.	The signal activity can be monitored from waveform.	Pass		
4	MM2S READ	read_test	Activates the Stream for Memory Mapped (MM2S) READ operations and initiates a Read Sequence to obtain the output on Stream. The Read Agent will transfer the data to the scoreboard, which has its internal memory pre-loaded with the same data (axi).	Configure the DMA's internal registers as following: 1. MM2S_DMCR = 32'h11001 2. MM2S_SA = cfg.SRC_ADDR	The DMA Should read same data from BRAM starting from the <b>SRC_ADDR</b> as the data	The scoreboard validates each byte received from the DMA against our memory model	Pass		

## 7. Coverage Scores



Groups Coverage Summary

Score	Inst Score
100	100

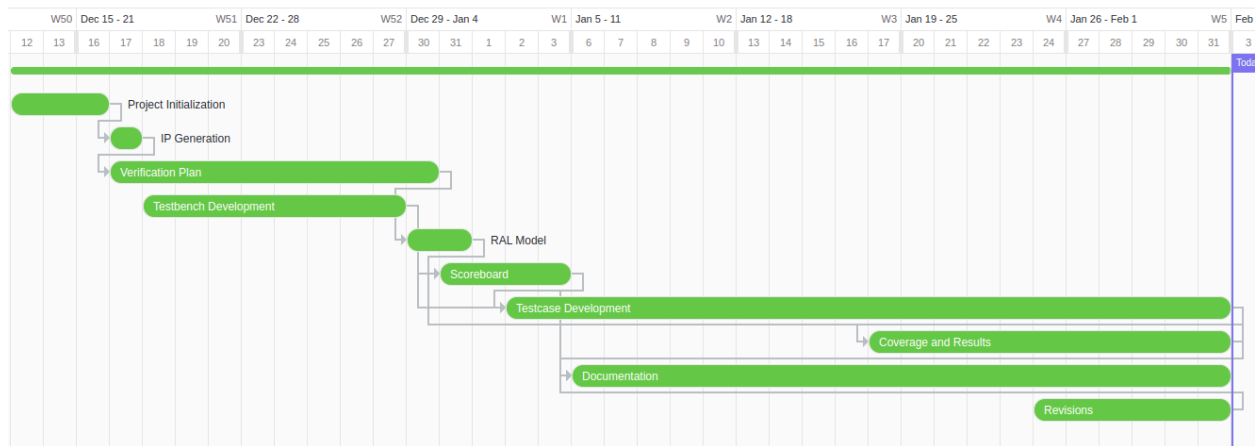
Total groups in report: 6

Name	Score	Num Instances	Avg Instances Score	Weight	Goal	Merge Instances	Get Inst Coverage	Per Instance	Auto Bin Max	Comment
\$unit_params_pkg_sv_891442008:axi_lite_coverage:cg_read_channel	100	1	100	1	100	0	0	0	64	
\$unit_params_pkg_sv_891442008:axi_lite_coverage:cg_write_channel	100	1	100	1	100	0	0	0	64	
\$unit_params_pkg_sv_891442008:axis_read_coverage:cg_axis_read	100	1	100	1	100	0	0	0	64	
\$unit_params_pkg_sv_891442008:axis_read_coverage:cg_mm2s_introut	100	1	100	1	100	0	0	0	64	
\$unit_params_pkg_sv_891442008:axis_write_coverage:cg_axis_write	100	1	100	1	100	0	0	0	64	
\$unit_params_pkg_sv_891442008:s2mm_introut_coverage:cg_s2mm_introut	100	1	100	1	100	0	0	0	64	

## 8. Verification Challenges

- Synchronization between AXI and AXI-Stream interfaces.
- Read after Write Test (RAW).
- Scoreboard Checkers
- Writing Assertions
- Debugging random failures.

## 9. Timeline



## 10. References:

- 10.1. [Git Repository](#)
- 10.2. [AXI Stream Specification](#)
- 10.3. [AXI4 and AXI-Lite Specification](#)
- 10.4. [AXI DMA Specification](#)