APB Master

The APB (Advanced Peripheral Bus) Master is a key component of the AMBA (Advanced Microcontroller Bus Architecture) family, designed to interface with low-power and low-bandwidth peripherals. As a master, its role is to initiate transactions such as read or write operations with APB slaves, which are typically peripheral devices like timers, UARTs, or GPIOs. The APB master ensures simplicity and efficiency in communication, adhering to the non-pipelined and sequential protocol characteristics of APB. Unlike AHB, the APB operates in a non-pipelined manner, meaning that each transaction (address phase and data phase) completes sequentially. This simplicity makes the APB suitable for peripherals that do not require high bandwidth or complex interconnects.

Design Aspects

• Support single word aligned transfers.

• Support two consecutive transactions without reaching idle state.

• Read and Write Operations: The master facilitates both read and write transactions, allowing it to transfer
data bidirectionally in compliance with the APB protocol.

• Low Power Consumption: Suitable for low-bandwidth, peripheral devices.

• Simple Protocol: No burst transfers; only single-cycle read/write operations.

• Easeof Integration: Commonly used for interfacing with simple peripherals like UART, timers, and GPIOs.

• Support for Multiple Peripherals: Allows multiple slaves to be connected via a simple decoder.

Data Transfer Workflow

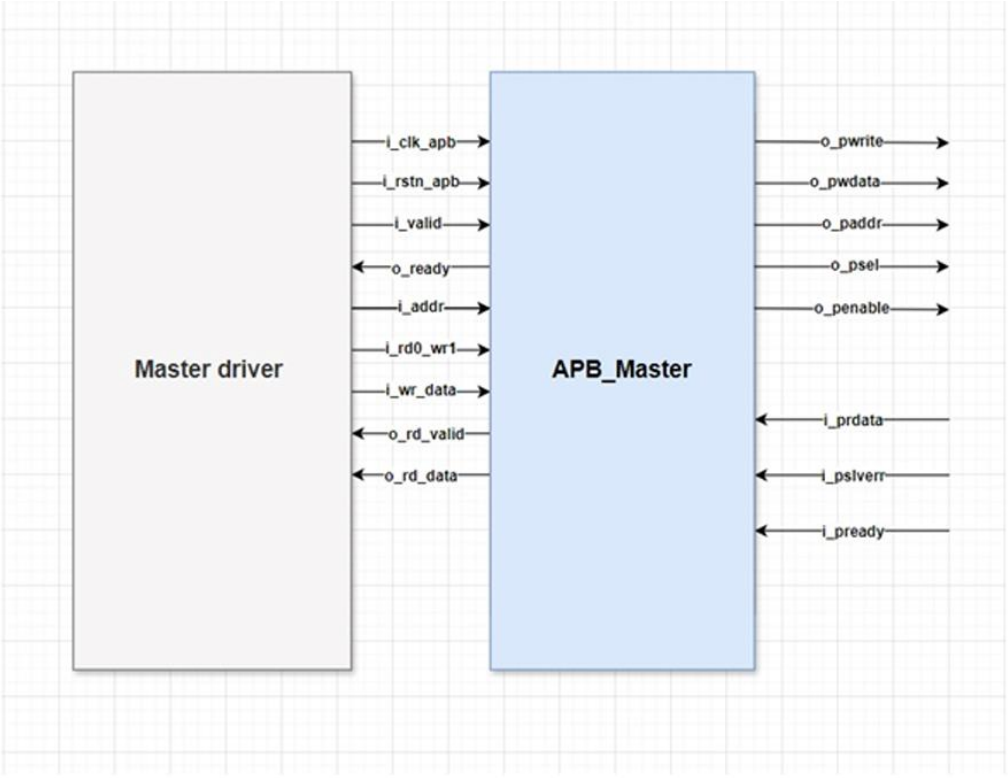
1. Driver Initiates Transfer

- Sets address (`i_addr`), command (`i_rd0_wr1`), and data (`i_wr_data` for writes).
- Asserts `i_valid` and waits for `o_ready` (acknowledgment from APB master).

2. Master Processes Request

- Buffers signals and starts APB protocol:
  - **Setup Phase**: Asserts `PSEL`, `PADDR`, `PWRITE` (keeps `PENABLE=0`).
  - **Access Phase**: Asserts `PENABLE` (next cycle) for data transfer:
    - **Write**: Sends data via `PWDATA`.
    - **Read**: Receives data via `PRDATA`.
- Slave completes transfer by asserting `PREADY` (inserts wait states if busy).

Architecture



APB Master UVM

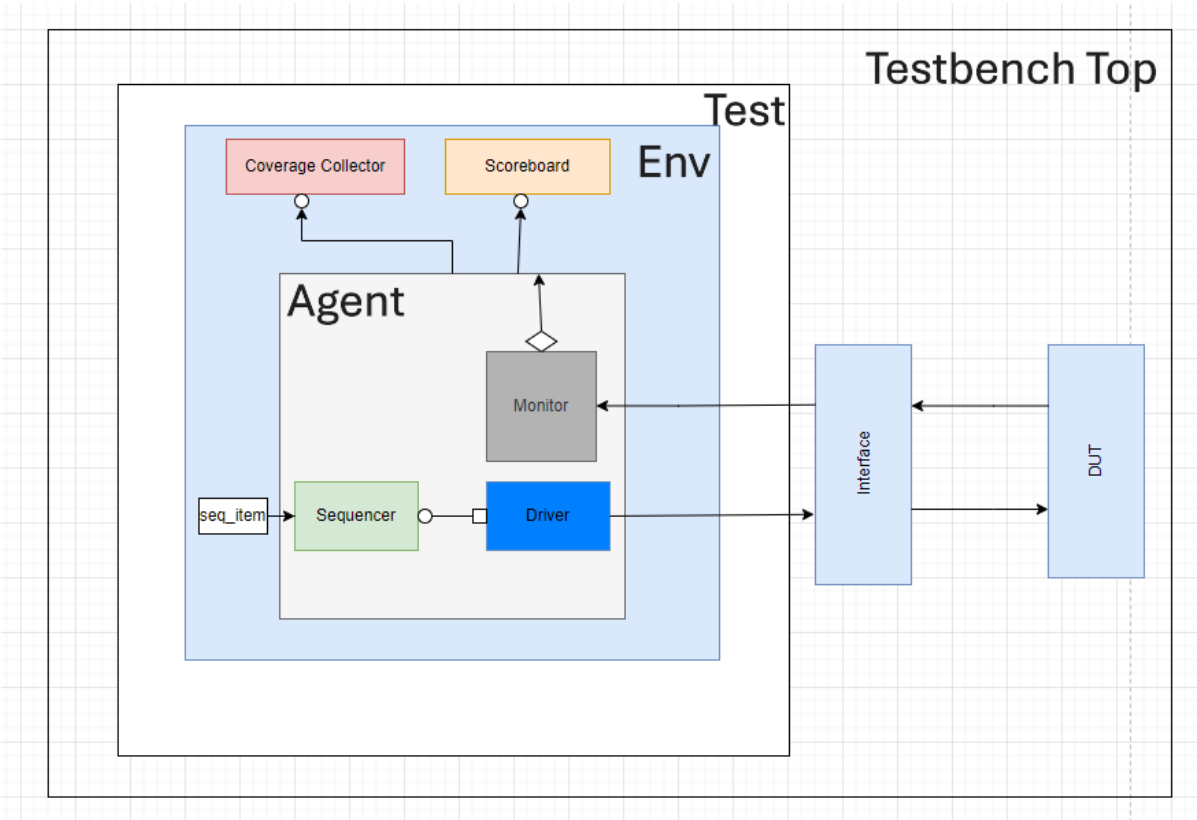
UVM architecture

UVM Component Role and Interface

This UVM component represents an **APB Slave Emulator** that interacts with and verifies the behavior of the **APB Master DUT**.

Key Details:

1. **Verification Setup:**
   - The DUT is an **APB Master**.
2. **Signal Flow:**
   - The APB Master (DUT) drives signals such as:
     - `o_psel`, `o_penable`, `o_pwrite`, `o_paddr`, and `o_pwdata`.
   - The UVM Slave emulates responses via signals:
     - `i_pready`, `i_prdata`, and `i_pslverr`.
3. **Handshaking Signals:**
   - The master initiates transactions using signals like `i_valid` and expects acknowledgments via signals like `o_ready`.



Verification Goals

The following verification goals have to met

1. **Protocol Compliance**
    a. Ensure the APB master adheres to the AMBA APB protocol specifications.
    b. Validate that signal transitions and timing relationships conform to the protocol.
2. **Transaction Generation**
    a. Verify the master initiates and completes read and write transactions correctly.
3. **Signal Handshakes**
    a. Confirm the master generates valid/ready handshakes (PSEL, PENABLE, and PREADY) appropriately.
    b. Ensure the master properly waits for PREADY before proceeding to the next transaction.
4. **Data Integrity**
    a. Verify the correctness of data sent on the PWDATA bus during write operations.
    b. Validate that the data received on PRDATA matches expectations during read operations.
5. **Corner Cases**
    a. Test boundary conditions such as maximum address width and maximum data bus width.
    b. Verify the master's behavior when multiple back-to-back transactions are initiated.
6. **Functional Coverage**
    a. Ensure comprehensive coverage of read and write operations for different data sizes.
    b. Track scenarios involving all valid signal combinations and transaction sequences.
7. **Stress Testing**
    a. Test the master under a high load of transactions to ensure stability and robustness.

Verification Strategies

To achieve the above verification goals we have to follow this verification strategies which are

a. **UVM Components**: Use UVM components like sequences, sequence items, and drivers to create robust transaction generation. You can define various sequences for read and write.
b. **Randomization**: Utilize UVM's built-in randomization features to generate diverse transaction patterns and corner cases to stress-test the interface.
c. **Functional Coverage**: Implement functional coverage to track which aspects of the APB protocol have been exercised, helping to identify untested scenarios.
d. **Assertions**: Use SystemVerilog assertions (SVAs) to enforce protocol rules and check for violations in real-time during simulation.

Coverage Model

Functional Coverage Description

Functional coverage is designed to ensure that all possible scenarios in the APB master design are thoroughly verified. It captures the behavior of the design under various conditions and checks the protocol's compliance. Below is an explanation of the key coverage points:

1. *Input Signals Coverage*
   - Tracks when the master receives valid signals to initiate transactions.
   - Ensures differentiation between read and write commands (***`i_rd0_wr1`***) is thoroughly tested.
   - Validates readiness (***`i_valid`***) before executing any operation.
2. *Output Signals Coverage*
   - Monitors valid read data signals (***`o_rd_valid`***) and the master's readiness to accept transactions (***`o_ready`***).
   - Tracks critical APB protocol signals:
     - Selection signal (***`o_psel`***): Ensures the correct peripheral is addressed.
     - Enable signal (***`o_penable`***): Validates the timing and assertion of the enable phase.
     - Write signal (***`o_pwrite`***): Checks whether write operations are correctly flagged.
3. *Handshake and Error Signals Coverage*
   - Monitors pready (***`i_pready`***): Ensures the master properly waits for peripheral acknowledgment.
4. *Cross-Coverage*
   - Combines key coverpoints to validate interactions between different signals.
   - Examples:
     - **`Mast-in-rd0-wr1`** crossed with **`mast_in_valid`** ensures all valid read/write operations are tested.
     - Crosses between ***`o_psel`***, ***`o_penable`***, and ***`o_pwrite`*** validate the proper sequence and timing of protocol operations.

Assertions

Assertions Table

Name	Description	Coverage
P1	**Asserts that the state transitions correctly from** IDLE **to** SETUP **when** i_valid **is asserted.**	**Ensures the state machine starts a transaction correctly.**
P2	**Asserts that the state transitions from** SETUP **to** ACCESS**.**	**Ensures the state machine moves to the ACCESS phase after SETUP.**
P3	**Asserts that the state transitions from** ACCESS **to** IDLE **when** i_pready **is high and** i_valid **is low.**	**Ensures the state machine returns to IDLE after completing a transaction.**
P4	**This assertion verifies that when an APB transfer completes during the ACCESS phase (with i_pready and i_valid high), the state machine transitions back to SETUP on the next clock edge.**	**Ensures correct APB protocol behavior after a completed transfer.**
P5	**Assert that o_rd_valid is asserted only for read transactions and when i_pready is high**	**Read Validity: o_rd_valid only asserts during read transactions and i_pready is high too**
P6	**Assert that o_rd_data is valid only when o_rd_valid is asserted**	**Ensures the correct data has been sampled**

Key Points in the Assertions Table

1. State Machine Transitions (p1-p4)

- **p1**: Ensures transition from `IDLE → SETUP` when `i_valid` is asserted.
- **p2**: Guarantees unconditional transition from `SETUP → ACCESS` (APB protocol requirement).
- **p3**: Verifies transition from `ACCESS → IDLE` if:
  - Transfer completes (`i_pready=1`)
  - No new request (`i_valid=0`).
- **p4**: Verifies transition from `ACCESS → SETUP` if:
  - Transfer completes (`i_pready=1`)
  - New request is pending (`i_valid=1`).

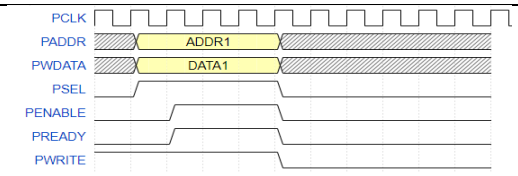
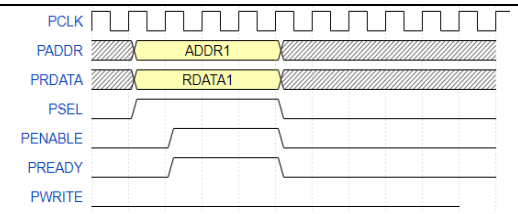
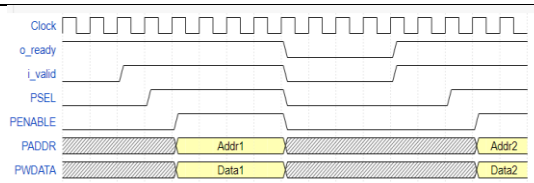
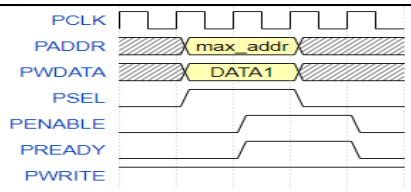
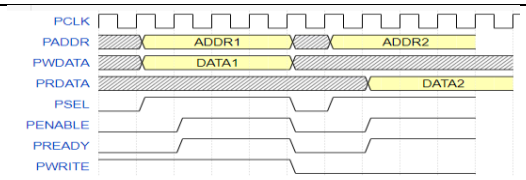
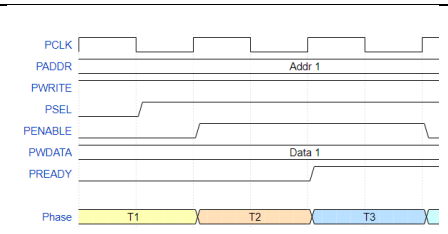
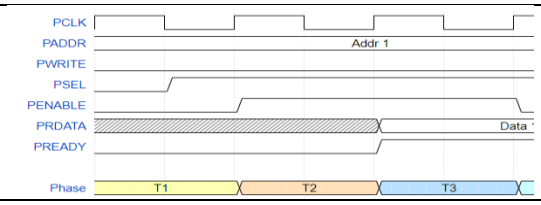
2. Read Data Validity (p5-p6)

- **p5**: Ensures `o_rd_valid` asserts **only** when:
  - In `ACCESS` phase
  - Peripheral is ready (`i_pready=1`)
  - It's a read transaction (`i_rd0_wr1_reg=0`).
- **p6**: Guarantees `o_rd_data` matches `i_prdata` **only** when `o_rd_valid=1` (data integrity).

Why These Assertions Matter

1. **Protocol Compliance**: Enforces correct APB state machine behavior.
2. **Data Integrity**: Prevents false read validations and ensures read data is stable.
3. **Deadlock Prevention**: Ensures the FSM never gets stuck (always returns to `IDLE` or `SETUP`).
4. **Bug Detection**: Catches:
   - Invalid state transitions
   - Read/write transaction mismatches
   - Peripheral handshake errors.

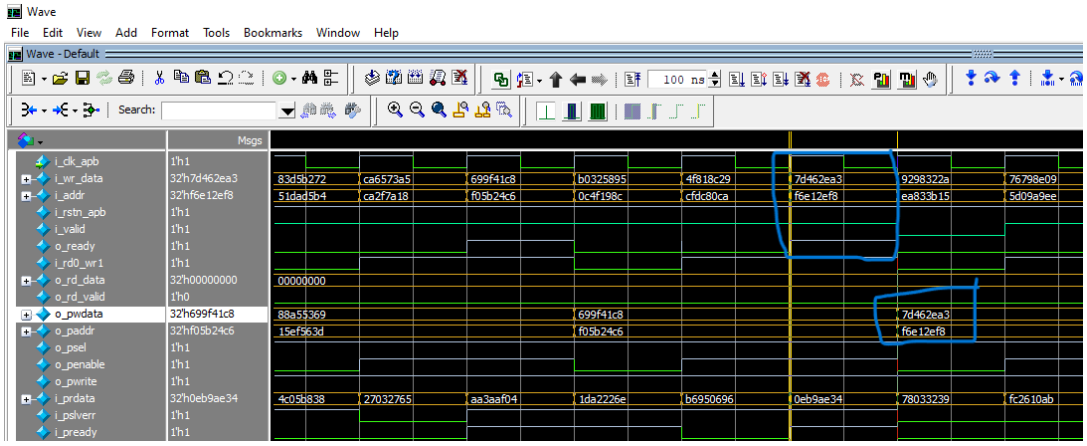
Test Case Table for APB Master Verification

Test Case ID	Description	Waveform	Expected Output	Comments
TC1	Basic Write Transaction		Data written successfully to the addressed peripheral.	Basic functional test.
TC2	Basic Read Transaction		Data read successfully from the addressed peripheral.	Ensures basic read functionality.
TC3	Handshake Validation		Write/read operation initiates only when ***o_ready*** is high.	Tests handshake mechanism (built in for our SOC).
TC4	Corner Case: Maximum Address Range		Data is written/read correctly at the maximum address range.	Boundary condition testing.
TC5	Back-to-Back Transactions		Transactions complete without timing issues or data corruption.	Validates sequential operations.
TC6	Write with wait states		Write transaction stalls until `PREADY` is asserted.	Validates behavior under handshake failure.
TC7	Read with wait states		Read transaction stalls until `PREADY` is asserted.	Validates behavior under handshake failure.

Simulation waveforms:

1. Simple write data transaction

The APB master ensures that write data and address signals are accurately propagated to the output during write operations.



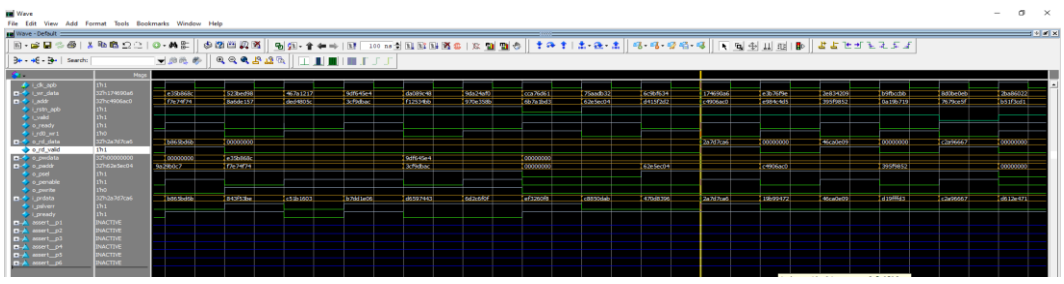
2. Back to back Transaction(read and write)

The waveform below demonstrates the **handshake mechanism** for a **back to back transactions** in our custom APB design. The key signals involved are:

- **PSEL**: Asserted by the master to select the slave.
- **PENABLE**: Asserted by the master during the access phase.
- **PREADY**: Asserted by the slave to complete the transaction.
- **o_ready**: A **custom signal** in our design, indicating that the master is ready to initiate a transaction.

For a successful write transaction:

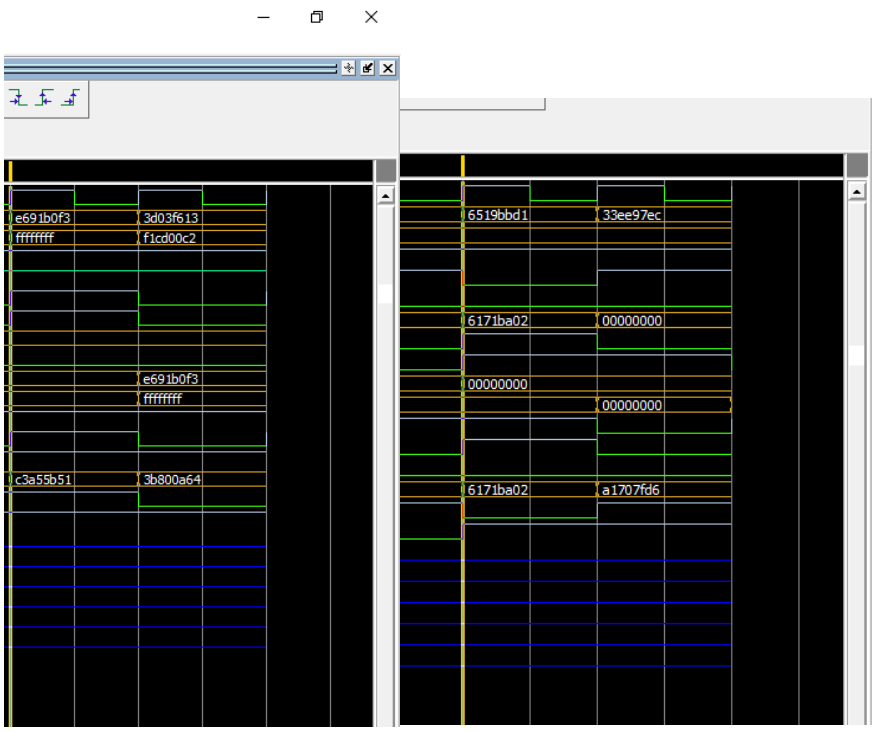
1. The master asserts **o_ready** to indicate it is ready.
2. The master then asserts **PSEL** and **PENABLE**, following the standard APB protocol.
3. The slave responds with **PREADY**, completing the transaction.



3. Max address was sampled right

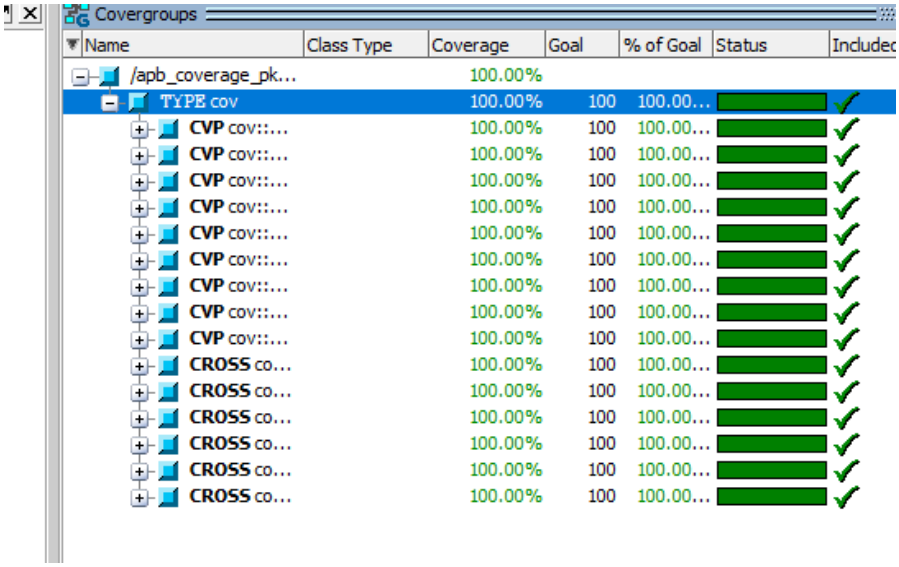
- The waveform below demonstrates Confirm that the **maximum address** (MAX_ADDR) is correctly handled by the APB master for both read and write operations

.



Verification Results and Reports

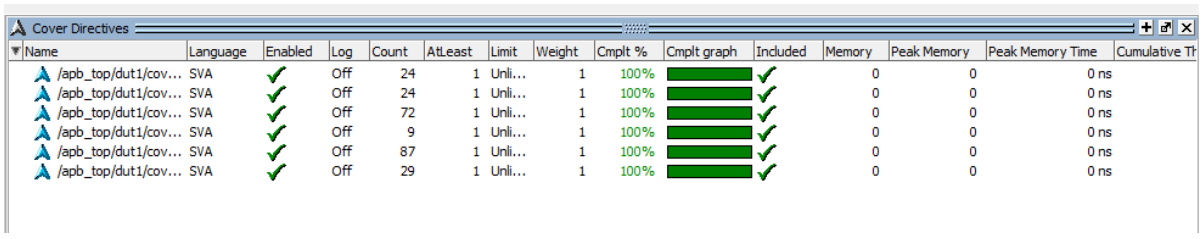
1. Functional coverage report



2. UVM report

```
# UVM_INFO apb_scoreboard_pkg.sv(48) @ 2585: uvm_test_top.env.sb [COMPARE] Transactions match
# UVM_INFO apb_scoreboard_pkg.sv(48) @ 2595: uvm_test_top.env.sb [COMPARE] Transactions match
# UVM_INFO apb_scoreboard_pkg.sv(48) @ 2605: uvm_test_top.env.sb [COMPARE] Transactions match
# UVM_INFO apb_scoreboard_pkg.sv(48) @ 2615: uvm_test_top.env.sb [COMPARE] Transactions match
# UVM_INFO apb_scoreboard_pkg.sv(48) @ 2625: uvm_test_top.env.sb [COMPARE] Transactions match
# UVM_INFO apb_scoreboard_pkg.sv(48) @ 2635: uvm_test_top.env.sb [COMPARE] Transactions match
# UVM_INFO apb_scoreboard_pkg.sv(48) @ 2645: uvm_test_top.env.sb [COMPARE] Transactions match
# UVM_INFO apb_scoreboard_pkg.sv(48) @ 2655: uvm_test_top.env.sb [COMPARE] Transactions match
# UVM_INFO apb_scoreboard_pkg.sv(48) @ 2665: uvm_test_top.env.sb [COMPARE] Transactions match
# UVM_INFO apb_scoreboard_pkg.sv(48) @ 2675: uvm_test_top.env.sb [COMPARE] Transactions match
# UVM_INFO apb_scoreboard_pkg.sv(48) @ 2685: uvm_test_top.env.sb [COMPARE] Transactions match
# UVM_INFO apb_scoreboard_pkg.sv(48) @ 2695: uvm_test_top.env.sb [COMPARE] Transactions match
# UVM_INFO apb_test_pkg.sv(60) @ 2695: uvm_test_top [run_phase]  END MASTER AGENT.
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 2695: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO apb_scoreboard_pkg.sv(56) @ 2695: uvm_test_top.env.sb [report_phase] Total successful transactions: 270, Failed transactions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :  280
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [COMPARE]   270
# [MASTER Agent]     3
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     1
# [run_phase]     2
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 2695 ns  Iteration: 61  Instance: /apb_top
```

3. Assertions coverage



Every assertion passed successfully