

# APB Slave

The APB (Advanced Peripheral Bus) slave is a key component of the AMBA (Advanced Microcontroller Bus Architecture) family, designed to interface with low-power and low-bandwidth peripherals. As a slave, its role is to respond to the master transactions such as read or write operations. The APB slave 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.
- Ease of 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. **Idle:** Waits for `PSEL=1` (selected by master).
2. **Setup:** Latches address (`PADDR`), direction (`PWRITE`), and write data (`PWDATA` if write).
3. **Access:**

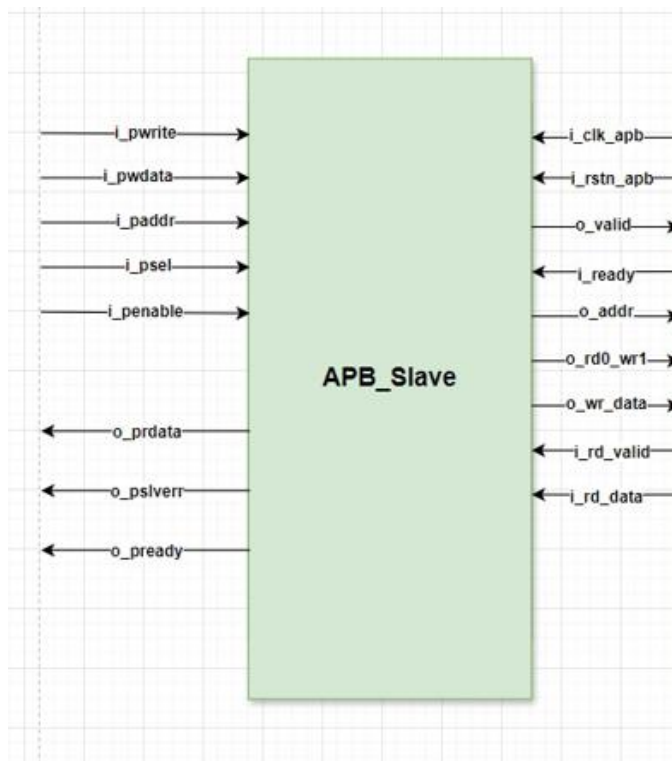
If `PENABLE=1` and slave is ready (`PREADY=1`):

- **Write:** Stores `PWDATA`.
- **Read:** Outputs data on `PRDATA`.

If busy, keeps `PREADY=0` (wait states).

4. **Complete:** Returns to idle (`PSEL=0`).

## Architecture



# APB Slave UVM

## UVM architecture

### *UVM Component Role and Interface*

This UVM Component Represents an **APB Slave DUT** That Interacts with and is Verified by the Behavior of the **APB Master**.

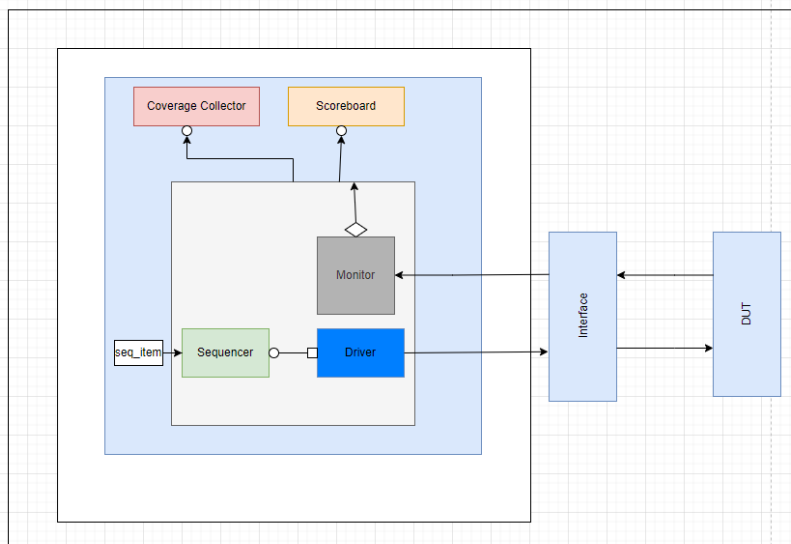
### *Key Details:*

1. **Verification Setup:**
  - The DUT is an **APB Slave**.
  - The UVM environment drives the behavior of an APB Master to validate the APB Slave's functionality.
2. **Signal Flow:**
  - The APB Master drives signals such as:
    - **o\_psel**, **o\_penable**, **o\_pwrite**, **o\_paddr**, and **o\_pwdata**.
  - The APB Slave DUT responds via signals:
    - **i\_pready**, **i\_prdata**, and **i\_pslverr**.
3. **Handshaking Signals:**
  - **The APB Master** initiates transactions using **i\_valid**.
  - **The APB Slave** acknowledges readiness and completes transactions using **o\_ready**.

---

### *Role in Verification:*

The UVM environment ensures the APB Slave DUT's compliance with the APB protocol by verifying its ability to handle read/write transactions, maintain proper signal timing, and correctly implement handshaking mechanisms.



## Verification Goals

The following verification goals have to met

1. **Protocol Compliance**
  - a. Ensure adherence to AMBA APB specifications, including signal transitions (PSEL, PENABLE, PREADY) and timing relationships.
2. **Transaction Handling**
  - a. Verify correct handling of read and write transactions.
  - b. Ensure smooth state transitions (IDLE, READ, WRITE) and back-to-back transaction processing.
3. **Signal Handshakes**
  - a. Validate readiness only after transaction completion and correct waiting for PENABLE.
  - 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 slave's behavior when multiple back-to-back transactions are received.
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 slave under a high load of transactions to ensure stability and robustness.
  - b. Validate the slave's ability to handle simultaneous read and write transactions efficiently.

## 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 slave 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 slave receives valid inputs to initiate transactions.
- Ensures differentiation between read (`i_pwrite = 0`) and write (`i_pwrite = 1`) operations is tested thoroughly.
- Validates readiness (`i_ready`) and valid read data (`i_rd_valid`) signals before processing.

#### *2. Output Signals Coverage*

- Monitors transaction validity (`o_valid`) for all read and write operations.
- Tracks key APB protocol signals:
  - **Address signal (`o_addr`)**: Ensures the slave responds to correct address inputs.
  - **Enable signal (`o_penable`)**: Verifies proper assertion during the enable phase.
  - **Write signal (`o_rd0_wr1`)**: Differentiates between read and write transactions.
  - **Response signal (`o_pready`)**: Validates readiness for transaction completion.
  - **Error signal (`o_pslverr`)**: Ensures error-free operation.

#### *3. Handshake and Timing Coverage*

- Monitors handshake between the master and slave:
  - **Selection signal (`i_psel`)**: Verifies correct peripheral addressing.
  - **Enable signal (`i_penable`)**: Validates timing consistency during enable phases.
- Ensures the slave waits for `i_pready` before transitioning states.

#### *4. Cross-Coverage*

Captures interactions between critical signals to validate functional relationships:

- **`i_ready` and `i_rd_valid`**: Ensures all valid handshake conditions are exercised.
- **`o_valid` and `o_rd0_wr1`**: Verifies behavior for all valid read and write scenarios.
- **`i_psel`, `i_penable`, and `i_pwrite`**: Ensures adherence to APB protocol sequences and timing.

# Assertions

**Assertion Table**

Name	Description	Coverage
<b>reset_behavior</b>	Asserts that the state machine resets to the IDLE state when the reset signal (i_rstn_apb) is asserted.	Ensures correct initialization of the state machine after reset.
<b>valid_transition_from_idle</b>	Asserts that the state transitions correctly from IDLE to either READ or WRITE when the APB select signal (i_psel) is asserted and the slave is ready (i_ready).	Ensures valid state transitions when a transaction request is detected.
<b>write_state_data_control</b>	Asserts that during the WRITE state, the control signal o_rd0_wr1 is set to 1 and the write data (o_wr_data) matches the input data (i_pwdata) when the APB enable signal (i_penable) is asserted.	Ensures correct execution of write operations with proper control and data.
<b>read_state_data_control</b>	Asserts that during the READ state, the control signal o_rd0_wr1 is set to 0 and the read data (o_prdata) matches the input read data (i_rd_data) when the read valid signal (i_rd_valid) is asserted.	Ensures correct execution of read operations with proper control and data.
<b>pready_assertion</b>	Asserts that the (o_pready) signal is set to 1 during the READ state when (i_rd_valid) and (i_penable) are asserted, or during the WRITE state when (i_penable) is asserted.	Ensures readiness is signaled only during valid transaction phases.
<b>valid_assertion_in_idle</b>	Asserts that the (o_valid) signal is set to 1 when the state is IDLE and the APB select signal (i_psel) is asserted.	Ensures (o_valid) is only asserted during IDLE state when a transaction starts.

## Key Points in the Assertions Table

### 1. State Machine Correctness:

- Assertions like **reset\_behavior**, **valid\_transition\_from\_idle**, and **pready\_assertion** ensure that the state machine transitions correctly between **IDLE**, **READ**, and **WRITE** states, which is fundamental to the APB protocol.
- Ensures the state machine responds appropriately to reset and transaction requests.

### 2. Signal Stability:

- Assertions like **write\_state\_data\_control** and **read\_state\_data\_control** ensure that critical signals (e.g., o\_wr\_data, o\_prdata, o\_rd0\_wr1) remain stable and correct during transactions.
- Prevents data corruption or protocol violations during read and write operations.

### 3. Handshaking:

- Assertions like **pready\_assertion** and **valid\_assertion\_in\_idle** ensure that handshaking signals (e.g., o\_pready, o\_valid) behave as expected during transactions.
- Ensures proper communication between the master and slave by validating readiness and transaction initiation signals.

### 4. Transaction Completion:

- Assertions like **pready\_assertion** ensure that the state machine correctly handles the completion of transactions by asserting o\_pready only during valid phases.
- Critical for continuous operation and preventing premature or incorrect transaction completion.

Test Case Table for APB Slave Verification

Test Case ID	Description	Waveform	Expected Output	Comments
TC001	Basic Write Transaction		o_wr_data matches i_pwdata, o_preedy=1 after transaction.	Verifies correct write operation.
TC002	Basic Read Transaction		o_prdata outputs valid data from i_rd_data, o_preedy=1 after transaction.	Verifies correct read operation.
TC003	No Peripheral Selection		No transaction occurs. o_preedy=1, o_valid=0.	Ensures slave remains idle without selection.
TC004	Idle State Behavior		FSM transitions to IDLE, o_preedy=0, and o_valid=0 after completion.	Validates return to IDLE state.
TC005	Back-to-Back Transactions		All transactions complete correctly without data loss or errors.	Verifies slave can handle consecutive transactions.
TC006	Wait State Insertion		o_preedy=0 during wait state; o_preedy=1 when ready.	Ensures slave inserts wait states properly.
TC007	Handshaking Signal Test		o_valid=1 only when slave has valid data; synchronizes with i_ready.	Verifies handshaking between master and slave.



# Simulation waveforms:

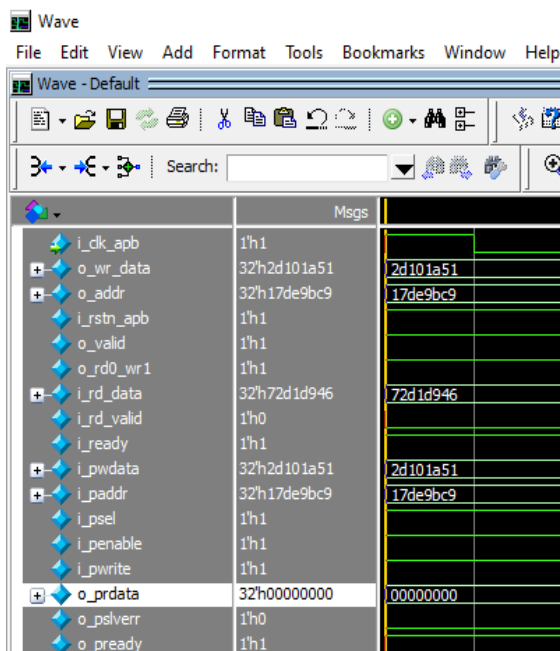
## 1.write data transaction

The waveform below demonstrates the handshake mechanism for a **write operation** 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.

*Steps for a Successful Write Transaction:*

1. The master asserts **o\_ready** to indicate it is ready to initiate a transaction.
2. The master asserts **PSEL** to select the slave and places the address (PADDR) and write data (PWRITE) on the bus.
3. The master asserts **PENABLE** during the access phase to validate the address and data.
4. The slave processes the write request and asserts **PREADY** to complete the transaction.



## 2. read data transaction

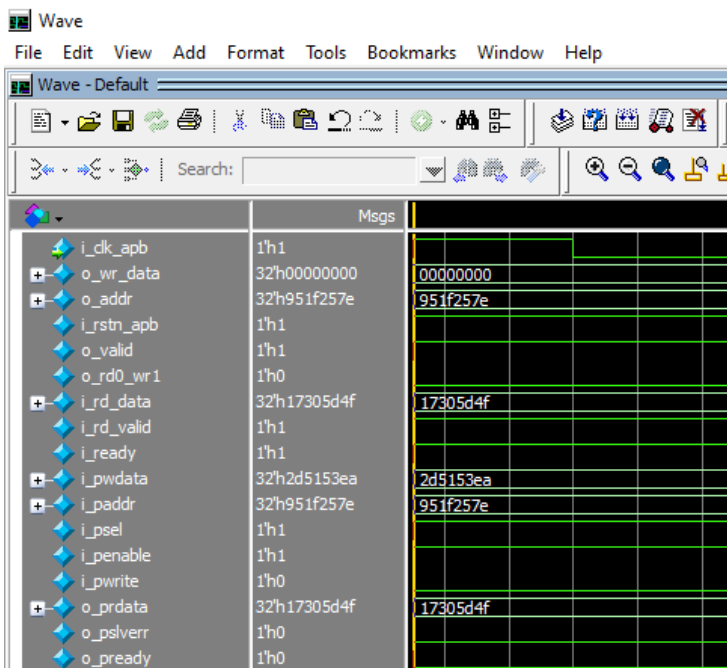
### Handshake Mechanism for Read Operation

The waveform below demonstrates the handshake mechanism for a **read operation** 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.
- **i\_rd\_valid**: A custom signal in our design, indicating that the read data is valid.

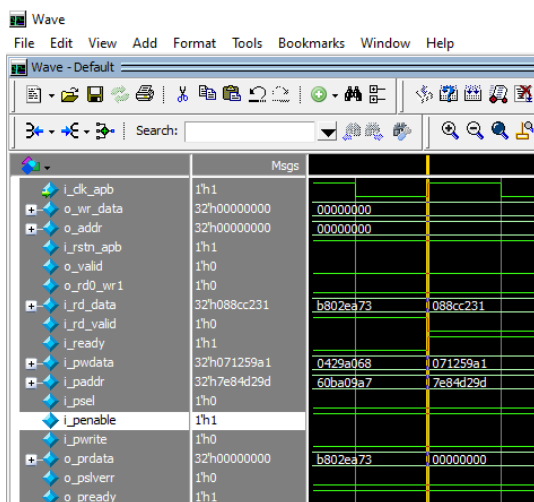
*Steps for a Successful Read Transaction:*

1. The master asserts **o\_ready** to indicate it is ready to initiate a transaction.
2. The master asserts **PSEL** to select the slave and places the address (PADDR) on the bus.
3. The master asserts **PENABLE** during the access phase to validate the address.
4. The slave processes the read request and asserts **PREADY** when the read data is ready.
5. The slave also asserts **i\_rd\_valid** to indicate that the read data (PRDATA) is valid.



### 3. When the Slave is not selected

- **No Transaction:** When PSEL is low, the slave is **not selected**, and no transaction (read or write) is in progress.
- **Bus Idle:** The bus remains idle, and all signals related to the transaction (e.g., PENABLE, PADDR, PWDATA, PRDATA) are either undefined or driven to default values.
- **Slave Inactive:** The slave does not perform any action and waits for PSEL to be asserted high to start a new transaction.



## Verification Results and Reports













## 1.Functional coverage report

[illegible]

## 2.UVM report

```
# Writes: 3467
# Reads: 3498
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO : 8975
# UVM_WARNING : 0
# UVM_ERROR : 0
# UVM_FATAL : 0
# ** Report counts by id
# [DRV] 2000
# [Questa UVM] 2
# [READ_XACT] 3498
# [REPORT] 1
# [RNTST] 1
# [TEST_DONE] 1
# [WRITE_XACT] 3467
# [run_phase] 2
# [slave Agent] 3
# ** Note: $finish : C:/questasim64_2021.1/win64/./verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
# Time: 150385 ns Iteration: 61 Instance: /apb_top
# 1
```

## 3.Assertions coverage

Cover Directives										
name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Include
 /apb_top/dut2/cov... SVA	SVA	✓	Off	6965	1	Unli...	1	100%		✓
 /apb_top/dut2/cov... SVA	SVA	✓	Off	2000	1	Unli...	1	100%		✓
 /apb_top/dut2/cov... SVA	SVA	✓	Off	1000	1	Unli...	1	100%		✓
 /apb_top/dut2/cov... SVA	SVA	✓	Off	1000	1	Unli...	1	100%		✓
 /apb_top/dut2/cov... SVA	SVA	✓	Off	2000	1	Unli...	1	100%		✓
 /apb_top/dut2/cov... SVA	SVA	✓	Off	5	1	Unli...	1	100%		✓

Every assertion passed successfully