Advanced High-performance Bus (AHB)

Revision History

| Revision | Description | Date | Modified By |
| --- | --- | --- | --- |
| 1.0 | Initial Document | 27/09/2023 | Pradip Prajapati |

Table of Contents

[Chapter 1: AHB Overview 1](#_Toc406486740)

[1.1 About the AHB protocol 1](#_Toc406486745)

[1.2 AHB revisions 1](#_Toc406486746)

[1.3 AMBA 3 AHB Protocol Specification v1.0 1](#_Toc406486747)

[Chapter 2: AHB features 3](#_Toc406486741)

[Chapter 3: AHB Verification Plan 4](#_Toc406486744)

[3.1 Feature Extraction 4](#_Toc406486745)

[3.2 Coverage Plan 4](#_Toc406486746)

[3.3 Checker Plan 4](#_Toc406486747)

[3.4 Verification Environment Development 4](#_Toc406486747)

[3.5 Test suite development 4](#_Toc406486750)

[3.5.1 Directed Testcases 4](#_Toc406486751)

[3.5.2 Random Testcases 5](#_Toc406486752)

[Chapter 4: AHB Verification Environment Development 6](#_Toc406486753)

[4.1 AHB Master-Slave Diagram 6](#_Toc406486754)

[4.2 AHB Master-Slave communication Diagam 6](#_Toc406486754)

[4.3 Verification Architecture 7](#_Toc406486754)

[4.4 AHB components 8](#_Toc406486755)

[4.4.1 Transcation Class 8](#_Toc406486756)

[4.4.2 Generator Class 8](#_Toc406486757)

[4.4.3 Driver 8](#_Toc406486758)

[4.4.4 Monitor 8](#_Toc406486759)

[4.4.5 Reference Model 8](#_Toc406486760)

[4.4.6 Score Board 8](#_Toc406486761)

[Chapter 5: Running Simulation 9](#_Toc406486772)

[Chapter 6: Closure Reports 10](#_Toc406486772)

List of Figures

[1.1 AHB Block Diagram 2](#_Toc406486745)

[4.1 AHB Master Block Diagram 6](#_Toc406486746)

[4.2 AHB Master Communication Block Diagram 6](#_Toc406486747)

[4.3 AHB verification architechture 7](#_Toc406486745)

List of Tables

# AHB Overview

## About the AHB protocol

The Advanced Peripheral Bus (AHB) is part of the Advanced Microcontroller Bus Architecture (AMBA) protocol family. It defines a low-cost interface that is optimized for minimal power consumption and reduced interface complexity.

The AHB protocol is not pipelined, use it to connect to low-bandwidth peripherals that do not require the high performance of the AXI protocol.

The AHB protocol relates a signal transition to the rising edge of the clock, to simplify the integration of AHB peripherals into any design flow. Every transfer takes at least two cycles.

The AHB can interface with:

• AMBA Advanced High-performance Bus (AHB)

• AMBA Advanced High-performance Bus Lite (AHB-Lite)

• AMBA Advanced Extensible Interface (AXI)

• AMBA Advanced Extensible Interface Lite (AXI4-Lite)

You can use it to access the programmable control registers of peripheral devices.

## AHB revisions

The AHB Specification Rev E, released in 1998, is now obsolete and is superseded by the following three revisions:

• AMBA 2 AHB Specification

• AMBA 3 AHB Protocol Specification v1.0

• AMBA AHB Protocol Specification v2.0.

## AMBA 3 AHB Protocol Specification v1.0

The AMBA 3 AHB Protocol Specification v1.0 defines the following additional functionality:

• Wait states.

• Error reporting.

The following interface signals support this functionality:

PREADY A ready signal to indicate completion of an AHB transfer.

PSLVERR An error signal to indicate the failure of a transfer.

**AHB Block Diagram**

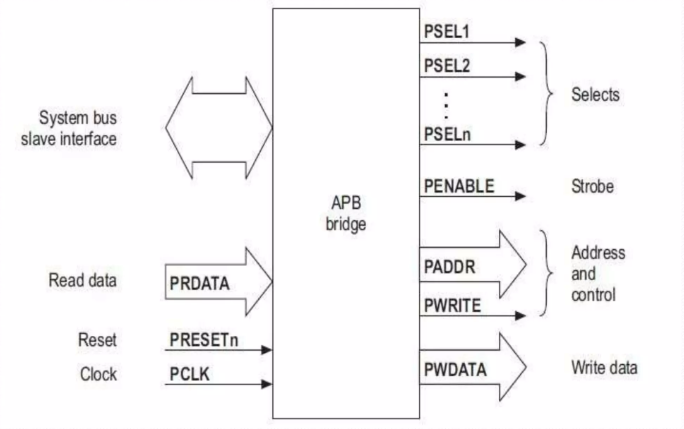


Figure 1.1: AHB Block Diagram

# AHB features

**The key features of the AHB are:**

* + Single edge protocol
  + Multiple bus masters and slaves
  + Write with Wait states and without wait states
  + Read with Wait states and without wait states
  + Low power consumption
  + Reduced complexity

# AHB Verification Plan

Verification involves studying the relevant specifications, extracting features from it that are to be tested, devising a strategy as to how these features are to be tested, developing a verification environment based on the strategy, writing testcases to cover all the scenarios and achieving 100% functional coverage figures.

## Feature Extraction

“Feature Extraction” involves listing out features to be tested from the specification. A feature list (spreadsheets) has been prepared using the AHB\_feature\_list document.

[AHB\_feature\_list.xlsx](APB_feature_list.xlsx)

The above spreadsheet lists out features to be tested in the corresponding specification, assigns a feature id to them and tells how the feature is to be tested (testcase name or checker task name).

## Coverage Plan

A functional coverage plan needs to be made based on the feature extraction document. This plan lists out the various combinations of stimuli that need to be generated for the proper verification of the UVC. This has been included in the feature extraction spreadsheet itself.

## Checker Plan

A checker plan needs to be made based on the feature extraction document. Implementations for the features marked as “checker” are elaborated here. We have included it in the feature extraction spreadsheet itself.

## Verification Environment Development

Our environment is based on SystemVerilog.

## Test suite development

### Directed Testcases

Testcases written to test specific areas of the UVC or to generate a specific kind or sequence of transactions is known as a directed testcase. These testcases are helpful in the initial and final stages of verification. In the initial stages, when neither the verification environment nor the UVC is matured, these testcases help in checking and correcting specific pieces of code in both the UVC and the verification environment. In the final stages, they are used to hit specific functional or code coverage areas.

### Random Testcases

Random testcases are written to test the UVC extensively. Random scenarios are generated based on constraints provided in the testcase. These testcases are run several times with different seed numbers to generate different scenarios to achieve more functional coverage figure.

# AHB Verification Environment Development

## AHB Master

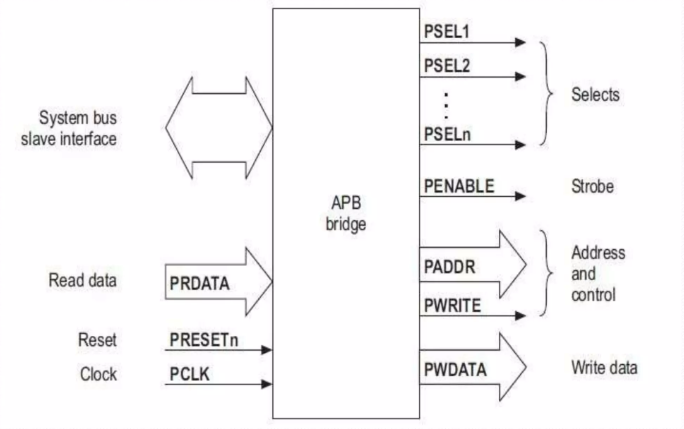


Figure 4.1: AHB Master Block Diagram

## AHB Master communication Block Diagram

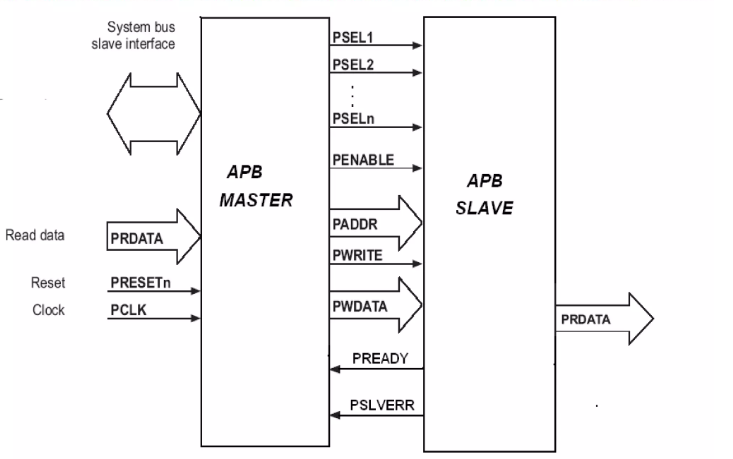


Figure 4.1: AHB Master Communication Block Diagram

## Verification Architecture

**TOP**

**TEST**

**ENV**

Generator

Scoreboard

Reference Model

Monitor

Driver

**DUT**

Figure 4.3: AHB Verification Architecture

## AHB Component

### Transaction Class

* It contains data members that represent the various signals or values that are part of the transaction.
* In transaction class of AHB, WR is define for selecting the mode of operation. In this 0 mode is define as READ state, 1 mode is defined as WRITE state. In the same write data and address is defined as random data. So, in the simulation write data and address will be randomized. For operation in AHB other signals also included.
* The transection class contains various behaviour and properties of a transection between various layered components.
* In our case it includes:

ADDR, PRDATA, WDATA, WR, Transf\_ctrl, PSLVERR, PREADY, PSEL, PENABLE, PADDR, PWRITE and PWDATA.

### Generator Class

* Responsible for stimulus/traffic (transaction items) generation and keep the same in mailbox which is further processed by driver.

i.e., stimulus generates randomly (preferred), through a file, hardcoded values, DPI

* In generator class, mb\_gen\_drv is mailbox for communicate with the environment. In this, virtual task run is taken for running the different test cases. In this one protected task is created name as put\_trans. this protected task will give permission to the child only for the extend of anything. Generator is used to generate stimuli for verification environment.
* By extending the generator class various test case scenarios implemented by randomization.
* Normal Read and Write without wait state
* Normal Read and Write with wait state
* Reset case
* Error cases

Process :

1. A mailbox (mb\_gen\_drv) of transaction class type is used to take generated data and provide it to driver.
2. In run method testcase writer will randomize the transaction class as per the testcase.
3. put\_trans() task is used to put the generated data in mailbox.
4. put\_trans() task is declared as protected but still child can access this method to put the data in mailbox.
5. new constructer with argument is used to connect the sub mailbox to main mailbox.
6. In generator class, mb\_gen\_drv is mailbox for communicate with the environment. In this, virtual task run is taken for running the different test cases. In this one protected task is created name as put\_trans this protected task will give permission to the child only for the extend of anything.

### Driver Class

* Who's responsible to takes transaction or sequence level activity(stimulus) coming from generator and convert it into the pin or system level activity.
* Driver drives/provides this pin or system level activity to bus (interface) as per protocol.
* It basically drives input data to design adhering to the protocol.
* In driver class, one mailbox mb\_gen\_drv is created for receiving the data. A virtual interface is connected with driver and at another side with the main interface. And the main interface is connected with the DUT.
* Driver simply drives the data which are generated to the DUT via interface.
* Send\_to\_dut function transfers the data from driver to DUT.
* Driver is connected to interface to provide the data to DUT so the virtual instance of interface of interface is taken(vif) from driver modport.
* Send\_to\_dut() task is used to drive the write and read data as per the clock.
* As per the WR ‘mode’ the mode is selected and as per the mode the respective write and read operation is done.

### Monitor Class

* Who's responsible to take pin or system level activity coming from bus (interface) and convert it into the transaction or sequence level activity.
* Monitor collect transaction or sequence level activity from bus (interface) as per the protocol.
* It basically sample/monitor interface data adhering to the protocol and send it to other component (i.e. scoreboard, reference model/predictor, coverage collector etc.)
* In monitor class, two mailbox are created. One mailbox is for sending the data from monitor to reference model (here mailbox name is mon\_rf) and one mailbox is for sending the data from monitor to score-board (here mailbox name is mon\_sb). One virtual interface is taken for transferring the data from interface to monitor.
* Monitor simply monitors pin level activities.
* It gets data from DUT and transfers to the reference model and scoreboard.
* In monitor class has two mailboxes. One mailbox is for sending the data from monitor to reference model (here mailbox name is mon\_ref) and one mailbox is for sending the data from monitor to score-board (here mailbox name is mon\_sb). One virtual interface is taken for sampling the output data of AHB the data from interface to monitor.
* The `forever` loop is indicating that monitor will continuously ready to sample anytime when data is available.
* Monitor() task is called at clock edge and data(write or read data) is sampled from interface and transferred to the reference model and scoreboard.

### Reference Model

* Reference model is a verification component where you wrote a logic to generate expected output (checker logics). (Predicting to output).
* Reference model is containing a queue array for replicating the data receiving from the monitor through the mailbox mon\_ref. it will create the expected data and send to the scoreboard.
* Reference model generates expected data for Design behaviour so basically it acts like DUT and generates the expected data for inputs. And send to the scoreboard.
* Reference model methods:

new method : It is used to construct mailboxes.

exp\_data method : it generates expected data.

run method : It gets data from monitor and send the actual data and expected data to the sb.

### Scoreboard

* Scoreboard is responsible to check whether your design output is correct or not.
* It's collects expected value from reference model, actual value from monitor and compare those values and log the status.
* Score-board contains two mail box as input. In those one is data transfer from monitor and another is from the reference model.
* Scoreboard get data from reference model and monitor than compares it and based on requirements and results it will give indication for pass and fail for perticular test.

The class contains the following methods:

* Scoreboard will take the data from the mon\_sb mailbox and also from ref\_sb mailbox and compare both data if it will match the “PASS” message is printed otherwise any mismatch is there then “FAIL” is printed.
* A run method is there in forever loop for taking the data from the both mailbox and pass as argument for compare method, compare method will compare the actual and expected data as print the “PASS” and “FAIL” message.

# Chapter 5: Running Simulation

# Chapter 6: Closure Report