AIM & INTRODUCTION

AIM:

This project deals with the class based verification of bridge between high speed AMBA AHB(Advanced High Performance bus) and low-power AMBA APB (Advanced Peripheral Bus) in UVM.

INTRODUCTION:

- Advanced High Performance bus (AHB):

  The AMBA AHB is for high-performance, high clock frequency system modules.

  The AHB acts as the high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macrocell functions. AHB is also specified to ensure ease of use in an efficient design flow using synthesis and automated test techniques

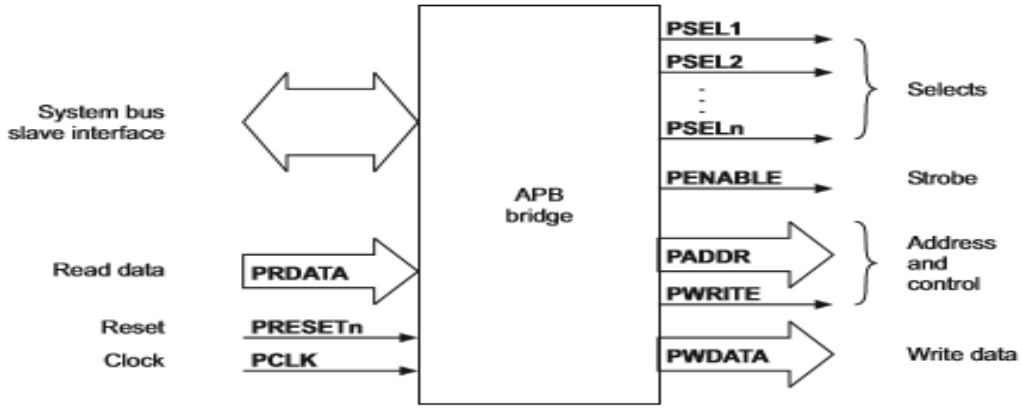
- Advanced Peripheral Bus(APB):

  The AMBA APB is for low-power peripherals.

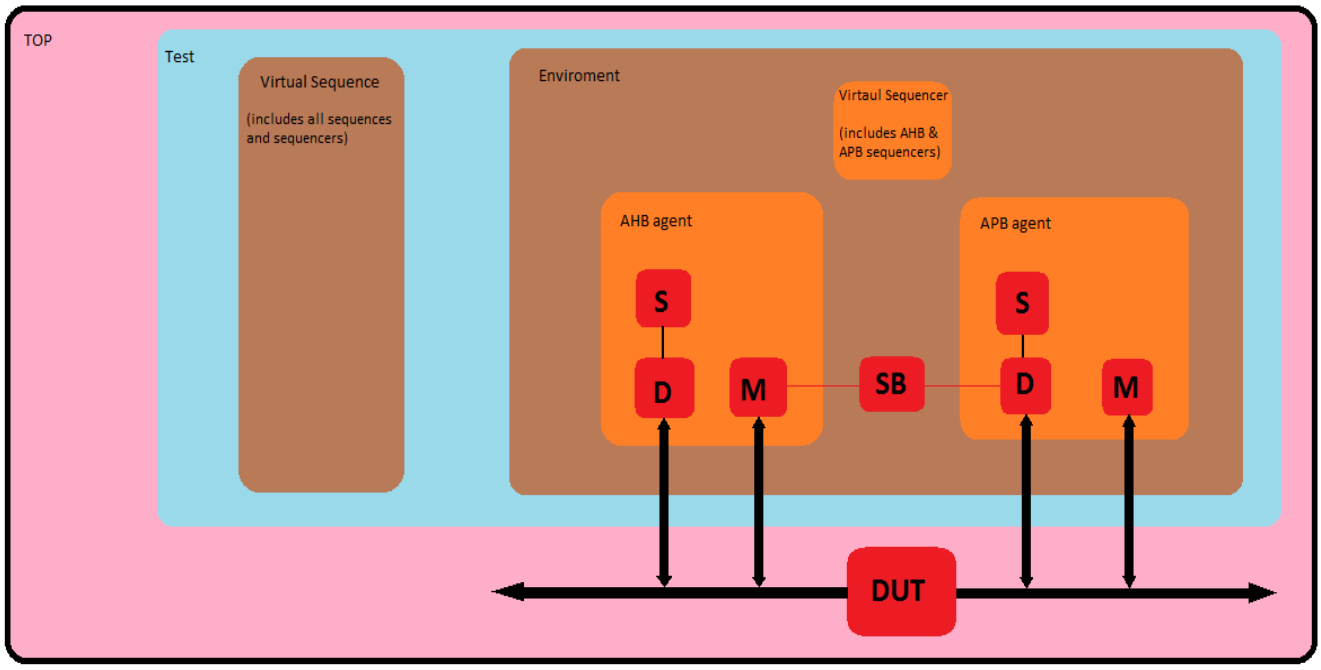
  AMBA APB is optimized for minimal power consumption and reduced interface complexity to support peripheral functions. APB can be used in conjunction with either version of the system bus.

- Functions of Bridge:

  1. Latches the address and holds it valid throughout the transfer.
  2. Decodes the address and generates a peripheral select, PSELx. Only one select signal can be active during a transfer.
  3. Drives the data onto the APB for a write transfer.
  4. Drives the APB data onto the system bus for a read transfer.
  5. Generates a timing strobe, PENABLE, for the transfer.



TESTBENCH ARCHITECTURE



TESTBENCH COMPONENTS

Driver

Driver is extended from uvm_driver and is parameterized with respective transaction class. The driver is connected to DUT through virtual interface with the help of config file. The driver class has TLM seq_item_port which is used to connect with the sequencer.

During the run_phase of the driver , data is collected from sequencer using get_next_item and acknowledgement is sent back by item_done.

- AHB driver logic: Wait till Hreadyout is high send control signals along with Haddr in the first clock cycle. In the next clock cycle check if Hreadyout is high and then send Hdata to DUT.
- APB driver logic: Wait till Penable is high and then drive Irdata when Pwrite is low.

Monitor

Monitor is extended from uvm_monitor and is connected to DUT through the virtual interface with the help of config file.

- AHB monitor logic: Wait till Htrans is 2 or 3 and then collect control signals from the interface . In the next clock cycle wait till Hreadyout is high and then collect data

signal.Create a copy of collected data and write into the tlm_analysis_fifo using 'write' method.

- APB monitor logic: Wait till Penable is high and check for Pwrite . If Pwrite is high then collect Pwdata and Paddr else collect Hrdata, Paddr and Irdata from the DUT. Send these collected items to second tlm_analysis_fifo in the scoreboard using 'write' method.

Sequencer

The sequencer is extended from uvm_sequnecer and is parameterized with respective transaction class. It is connected with the driver using seq_item_export.

Agent

Agent class is extended from uvm_agent . In the build_phase of agent, based on whether the agent is active or passive, driver, sequencer and monitor is built. In the connect_phase, connection between driver and sequencer is made.

Virtual sequencer

Virtual sequencer class is extended from uvm_sequencer and is parameterized with uvm_sequence_item. It consists handles of AHB and APB sequencers.

Environment

Environment class is extended from uvm_env. In the build_phase of environment AHB agent, APB agent , Virtual sequencer and Scoreboard is build. The connection between Scoreboard's tlm fifo and monitor's analysis port is made in the connect_phase.

Scoreboard

Scoreboard is extended from uvm_scoreboard and consists of 2 tlm fifos for collecting the items from AHB and APB monitor.

Get the data from AHB monitor and if Hwrite is high , push Hwdata and Haddr into locally declared queue. Get the data from APB monitor , simultaneously pop from queue and compare Hwdata with Pwdata and Haddr with Paddr.
If Hwrite is low, compare Hrdata with Irdata and Haddr with Paddr. Covergroup is included for signals which are to be tracked.

Virtual Sequence

Virtual sequence class is extended from uvm_sequence and is parameterized with uvm_sequence _item. It consists of handles of virtual sequencer , all sequencers and all the sequences. All the local sub-sequencers are made to point to sequencers inside virtual sequencer. This is possible by casting m_sequencer and virtual sequencer. Any sequence can be started using start method and it is started on the sequencer.

TESTCASE DETAILS

Hwrite, Hsize(constraint added so the value is between 0&2) are randomized and hence with each testcase different scenarios like write/read and different Hsize is covered.
These are the different testcase that were covered,

1. Single transfer
2. 4-beat wrapping burst.
3. 4-beat incrementing burst.
4. 8-beat wrapping burst.
5. 8-beat incrementing burst.
6. 16-beat wrapping burst.
7. 16-beat incrementing burst.

COVERAGE REPORT

Covergroup	Hits	Goal / At Least	Status
▼ TYPE */\package bridge_test_pkg\/scoreboard/bridge_coverage*	95.833%	100.000%	Uncovered
▼ INSTANCE *<UNNAMED1>*	95.833%	100.000%	Uncovered
▸ COVERPOINT *<UNNAMED1>::HSIZE*	100.000%	100.000%	Covered
▸ COVERPOINT *<UNNAMED1>::HBURST*	87.500%	100.000%	Uncovered
▸ COVERPOINT *<UNNAMED1>::HWRITE*	100.000%	100.000%	Covered