TITTLE: DEVELOPMENT AND VERIFICATION OF AHB2APB BRIDGE PROTOCOL USING UVM TECHNIQUE

AUTHOR: N.G.N.PRASAD,ASSISTANT PROFESSOR IN K.I.E.T COLLEGE,KORANGI

Abstract:
The AMBA AHB is for high-performance, high clock frequency system modules. The AHB acts as the high-performance backbone system bus. AHB supports the efficient connection of processors. The AMBA APB is optimized for low power consumption and interface reduced complexity to support peripheral functions. In this project functions of the AHB2APB Bridge protocol by writing the code in VERILOG and simulating it in XILINX ISE. In this project, we verify the all functions of Bridge protocol by writing verification code in UVM with different test cases. The code coverage and functional coverage and functional verification of the Bridge RTL design is 100% covered by using QUESTASIM.

Keywords: AHB, APB, QUESTASIM, XILINX ISE, AHB2APB Bridge, Verilog, UVM, Coverage, FPGA.

I. INTRODUCTION

The AHB bus protocol is designed to be used with a multiplexer interconnection scheme. Using this scheme all bus masters drive the address and control signal indicating the transfer to perform and the arbiter determines which master has its control signals and address routed to all of the slaves. The central decoder is also required to control the data read and signal response multiplexer, which selects the appropriate signals from the slaves that is involved in the transfer. The APB should be used to connected to any peripherals which are low bandwidth and do not need high performance of a pipelined bus interface.

The BUS Communication may be done in different ways.

(A) Transfer type: - Indicates type of the current transfer, which can be NON-SEQUENTIAL, SEQUENTIAL, IDLE or BUSY.

(B) Transfer direction: - When write HIGH this signal indicates a write transfer and when write LOW a read transfer.

(C) Transfer size: - Indicates this size of the transfer, which is typically byte (8-bit), half word (16-bit) or word (32-bit). The protocol allows larger transfer sizes up to a maximum of 1024 bits.

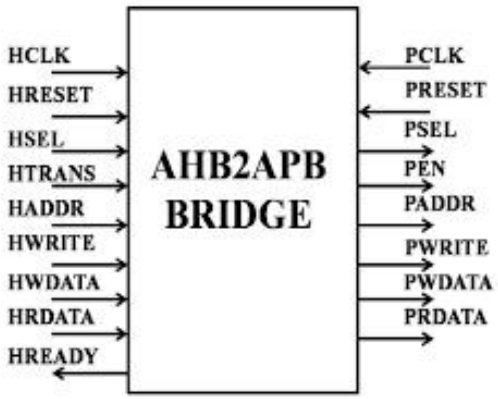
(D) Burst type: - Indicates if the transfer forms the part of a burst. Four and eight and sixteen beat bursts are supported and the burst may be either incrementing or wrapping.

II. AHB2APB BRIDGE

General architecture of AHB2APB Bridge consists of five main building blocks:

- AHB Master
- AHB2APB Bridge
- AHB Interface
- APB FSM Controller
- APB Interface

Architecture of AHB2APB Bridge

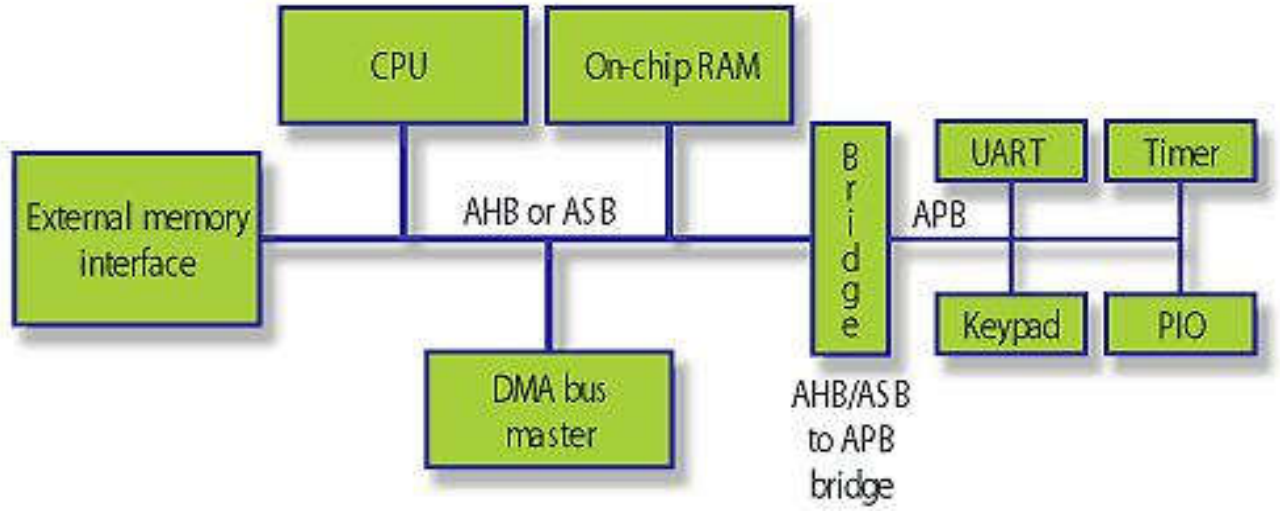


- AHB Master: The bus master is able to initiate read and write operations by providing an address and control information. One bus master is allowed to actively use the bus at any one time.

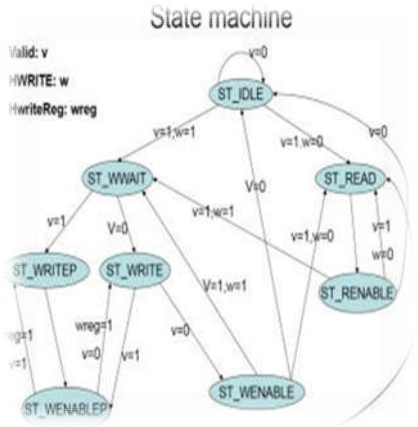
- AHB Arbiter: The bus arbiter ensures that one bus master at a time is allowed to initiate data transfers. Even though the arbitration protocol is fixed, any arbitration algorithm, such as highest priority and fair access can be implemented depending on the application requirements. An AHB would include one arbiter, although this would be trivial in single bus master systems.
- AHB decoder: The AHB decoder is used to decode address of each transfer and provide a select signal for the slave that is involved transfer single centralized decoder is required in all AHB implementations.
- APB Interface: A bus slave responds to a read and a write operation within a given address-space range. The bus slave signals back to the active master the success, failure and waiting of the data transfer address and data that received from the bridge suitably used for data transaction from bridge to this module and vice versa depending whether it is a write and a read operation. These modules contain block for P_CLK generation, which is distributed to AHB2APB bridge module, and the P_CLK generated obviously used in this module also.

    - AHB2APB Bridge: Out of all modules present, this module is simplest and also very larger. All the signals are taken as wire to interconnect the various modules present in the top module. In the module, all the three modules namely
    - AHB Master
    - AHB2APB bridge
    - APB interface

These modules are all instantiated used Positional assignments which is again simple compared naming assignment which is little tedious.



6. FSM Controller:

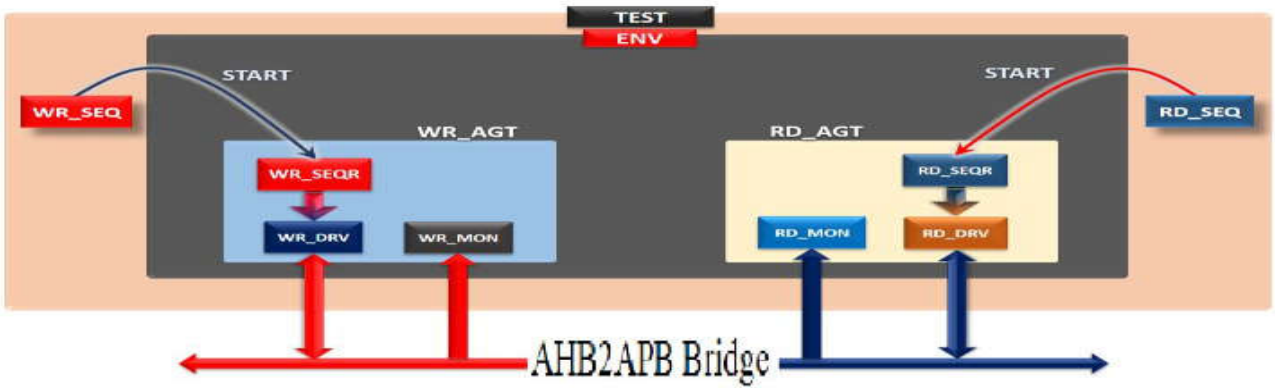


III. UVM (UNIVERSAL VERIFICATION METHODOLOGY)

The Universal Verification Methodology was introduced in December 2009, by the technical Subcommittee of Accellera. UVM use Open Verification Methodology and its foundation. Accellera released version UVM 1.0 EA on May 17th 2010. UVM Class Library provides the building blocks needed quickly develop well-constructed as well as reusable verification components and test environments. It uses system Verilog as language. All three of simulation vendors (Synopsys, Cadence and Mentor) supports UVM today which was not a case with other verification methodology. Today, more and more logic being integrated on the single chip so verification of it a very challenging task. More than 70 % of the time is spent on the verification of the chip. So, it is a need of an hour to have common verification methodology that provides base classes and framework to construct reusable verification environment. UVM provides that.

In this paper, all terminology related to UVM is introduced along with sample example. In first phase UVM components are introduced. In second phase, some of features related to UVM are introduced and final phase small environment is built using UVM from the scratch.

IV. UVM Testbench architecture



The following subsections describes the components of a verification environment.

- Data Item
- Driver
- Sequencer
- Monitor
- Agents
- Environment
- **Data Item:**

Data items are basically input to the device under test. All the transfers done between different verification components in UVM is done through transaction object. Network packets and instructions for processor are some examples of transactions. From top level test many data items are generate and applied to DUT so by intelligently randomizing the data items object we can check corner case and Maximize the coverage on the device under test.

- **Driver:**
  Driver as the name suggest, drive DUT signals. It basically receives the transaction object from sequencer and converts it to pin level activity. So, for example it can generate read and write signal, write address and data to be transferred. It is active part of the verification logic.
- **Sequencer:**

  Sequencer is the component which the sequence will run. The DUT needs to be applied a sequence of transaction to test its behaviour. So, sequence of transaction generated and it is applied to driver whenever it demands by sequencer.

- **Monitor:**

  A monitor is passive element of verification environment. It just samples the DUT signal from the interface but does not drive them. It collects the pin information, packages it in form of packet and then transfers it to scoreboard and components for coverage information.

- **Agent:**

  Agent is basically container. It contains driver, monitor and sequencer. The Driver and sequencer are connected as agent. Agent has two modes of operation: passive and active. In active mode drives the signal to DUT. So, driver and sequencer are instantiated active mode. In passive mode, it just samples the DUT signals does not drive them. So only monitor is instantiated passive mode. Normally there is one agent interface like AHB or APB.

- **Scoreboard:**

  Scoreboard is verification component that check the response from the DUT against the expected response. So, keeps track of how many times the response is matched with the expected response and how many times failed.

- **Environment:**

  Environment is the top of the test bench architecture. it will be contained one or more agents depend on the design. If more than one agents are there then it will connect in this component. Agents are also connected to another component like scoreboard in this component.

V. RESULTS AND DISCUSSION

Fig. Bridge top Schematic Diagram from Synthesis

The Bridge is carried out for the functional verification using the UVM techniques for both the read as well as write operation. The functional verification of the RTL design is the Bridge is yields the complete code and functional coverage.

Functional Verification of the Bridge Using UVM As verification methodology plays important phase in the circuit design. The read operation of the Bridge is carried out in XILINX for the RTL design and the verification methodologies are carried out using Questasim 10.2c. The design is carried out using in Verilog and the verification is carried out in UVM. The Bridge is set up as DUT the functional verification and the code coverage is obtained for 100%.

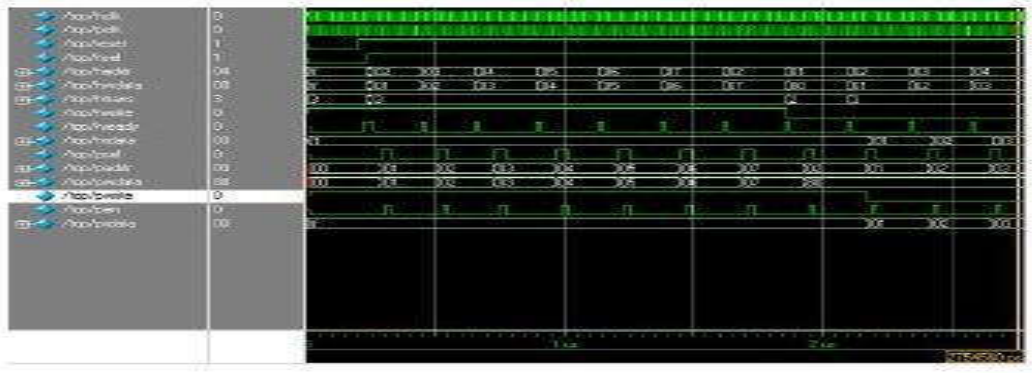
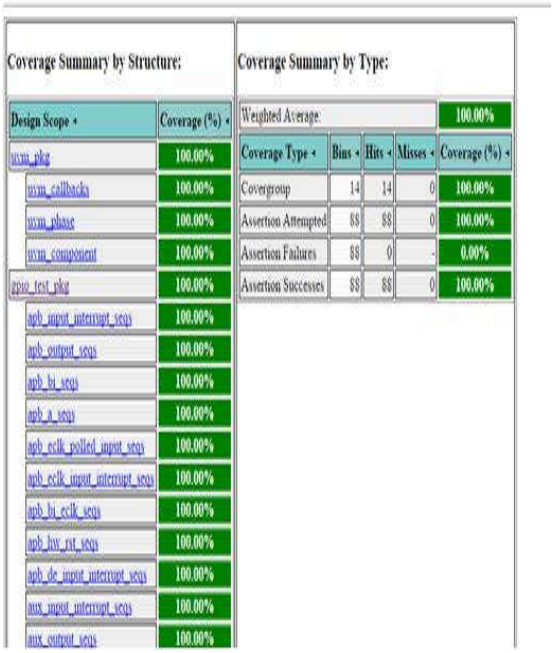


Fig. simulation showing Bridge Functional Verification for write operation

VI. COVERAGE REPORT



Coverage Summary by Structure:

Design Scope	Coverage (%)
uvm_pkg	100.00%
uvm_callbacks	100.00%
uvm_phase	100.00%
uvm_component	100.00%
gpio_test_pkg	100.00%
apb_input_interrupt_seqs	100.00%
apb_output_seqs	100.00%
apb_bi_seqs	100.00%
apb_a_seqs	100.00%
apb_eclk_polled_input_seqs	100.00%
apb_eclk_input_interrupt_seqs	100.00%
apb_bi_eclk_seqs	100.00%
apb_hw_rst_seqs	100.00%
apb_de_input_interrupt_seqs	100.00%
aux_input_interrupt_seqs	100.00%
aux_output_seqs	100.00%

Coverage Summary by Type:

Weighted Average:				100.00%
Coverage Type	Bins	Hits	Misses	Coverage (%)
Covergroup	14	14	0	100.00%
Assertion Attempted	88	88	0	100.00%
Assertion Failures	88	0	-	0.00%
Assertion Successes	88	88	0	100.00%

VII. CONCLUSION

In this we have designed and verified AHB2APB Bridge using Verilog and UVM techniques using Questasim. The code coverage is obtained for RTL design and 100% code coverage and functional coverage is extracted. The methodology provides the complete coverages of RTL design so as to acquire the fault free Protocol design of Bridge. So that can be implemented in real time system. This can be further implemented for the ASIC implementation and SOC Applications.

References:

[1] K.Cho et al, “Reusable Platform Design Methodology For SOC Integration And Verification”, Proceedings of ISOCC 2008, pp. I-78- I-81, Nov. 2008

[2] D.Gajski et al, “Essential Issues for IP Reuse”,

Proceedings of ASP-DAC, pp.37-42, Jan. 2000

[3] ARM Limited AMBA specification opencores.org

[4] Xilinx ISE Synthesis and Verification Design Guide

[5] Jaehoon Song, Student member, IEEE, Hyunbean Yi, Member,IEEE, Juhee Han, and Sungju Park, Member, IEEE,” An Efficient SOC Test Technique by Reusing On/Off Chip Bus Bridge”IEEE Transcactions on Circuits And Systems-I: Regular Papers, Vol,56, No.3, March2009.