He_Functional Verification of AXI4 Lite to AHB Bridge using UVM Technology

Electrical Engineering Department

PATEL, HARSHIL (MS Electrical Engineering)
SHANKAR, VIJAYKUMAR (MS Electrical Engineering)

Project Advisor: Dr. Lili He

Introduction

According to Moore's law, the number of transistors will be getting double every two years. This is a challenge for SoC domain, to implement millions of transistors on a single chip and maintain proper interoperability. Bus protocols provides smooth communication between internal components of the SoC. Hence to achieve this state, verification is required for the design, handling communication process in SoC.

This project is about performing functional verification using UVM technology for the AXI4 Lite to AHB bridge. The need of a bridge type connection reduces physical connectivity in a design up to a larger extent. This also eliminates the unwanted timing delays like propagation delay.

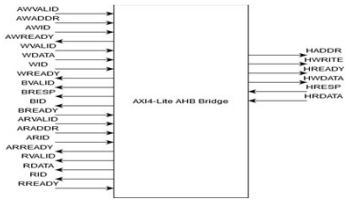


Figure 1. AXI4 Lite-AHB bridge port list

Figure 1. displays the interface signals connectivity between the AXI4 Lite and AHB bridge. All the signals defines the read and write functioning of the data.

Methodology

AXI4 Lite-AHB Bridge block diagram

In figure 2. the AXI4 Lite to AHB bridge owns an AXI4 lite transaction slave port and an AHB transaction master port. The AXI4 Lite slave port communicates with the AXI4 Lite master testbench through AXI channels. The data is now sent over to the AHB slave memory through the bridge.

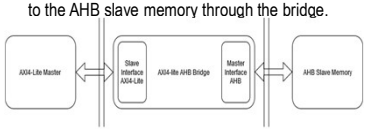


Figure 2. Block diagram

Slave interface module

The data transmission is taking place between a single master and a single slave interface. The AXI4 Lite master testbench is connected with a write buffer FIFO. This master will be driving the DUT with the reads and writes signals at VALID HIGH. as shown in figure 3. Moreover, the slave interface will be responding to the testbench unless the FIFO is full.

Methodology

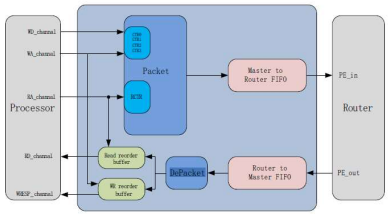


Figure 3. Slave interface module

The write transaction packet in figure 4. describes the size of address and data being written in the FIFO.

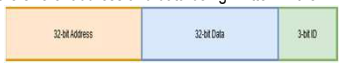


Figure 4. Write transaction packet

Master interface module

The master interface here will receive the unpacked data sent from the write FIFO buffer. This information will be further sent to the AHB slave memory through a round robin Arbiter as shown in figure 5.

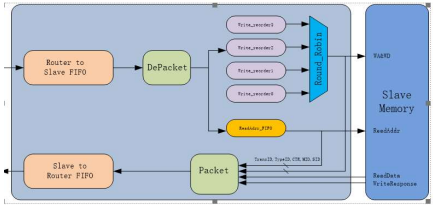


Figure 5. Master interface module

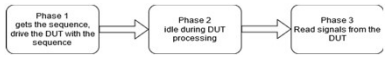
UVM Testbench hierarchy

UVM testbench hierarchy in a bottom-up fashion is built before the simulation starts. This hierarchy will define all the signals connected with different UVM components in the test module.

```
UVM_INFO @ 0: reporter [RNTST] Running test axilahb_test...
UVM_INFO /home/eerta/uvm-1.2/src/base/uvm_root.svh(579) @ 0: report
----------------------------------------------------------------------
Name                        Type                        Size  Value
----------------------------------------------------------------------
uvm_test_top                axilahb_test                -     @340
  env                       axilahb_environment         -     @353
    AWReady_verify          AWReady_verify              -     @306
      AWReady_iport_s       uvm_tlm_analysis_fifo #(T)  -     @395
        analysis_export     uvm_analysis_imp            -     @444
        get_ap              uvm_analysis_port           -     @434
        get_peek_export     uvm_get_peek_imp            -     @414
        put_ap              uvm_analysis_port           -     @424
        put_export          uvm_put_imp                 -     @404
    Active_Agent            axilahb_active_agent        -     @368
      driver                axilahb_driver              -     @523
        rsp_port            uvm_analysis_port           -     @542
        seq_item_port       uvm_seq_item_pull_port      -     @532
      monitor               axilahb_active_monitor      -     @689
        AWReady_iport       uvm_analysis_port           -     @698
        WReady_iport        uvm_analysis_port           -     @708
      sequencer             axilahb_seqr                -     @552
        rsp_export          uvm_analysis_export         -     @561
        seq_item_export     uvm_seq_item_pull_imp       -     @679
        arbitration_queue   array                       0     -
        lock_queue          array                       0     -
        num_last_reqs       integral                    32    'd1
        num_last_rsps       integral                    32    'd1
    Passive_agent           axilahb_passive_agent       -     @377
      pass_mon              axilahb_passive_monitor     -     @726
    WReady_verify           WReady_verify               -     @454
      WReady_iport_s        uvm_tlm_analysis_fifo #(T)  -     @463
        analysis_export     uvm_analysis_imp            -     @512
        get_ap              uvm_analysis_port           -     @502
        get_peek_export     uvm_get_peek_imp            -     @482
        put_ap              uvm_analysis_port           -     @492
        put_export          uvm_put_imp                 -     @472
----------------------------------------------------------------------
```

Figure 6. UVM topology

The driver drives the DUT through the sequences in following manner:



Analysis and Results

Figure 7. demonstrates how data is loaded in DUT for a write transaction. Both AWVALID and AWREADY are 1 for writing address channel in DUT, similarly for writing data channel, WVALID and WREADY are 1.

```
        (Specify +UVM_NO_RELNOTES to turn off this notice)
UVM_INFO @ 0: reporter [RNTST] Running test axilahb_test...
Resetting
----------------------------------------------------------------
Starting Sequence
Write Addr channel
Driving Write address Sequence to dut
AWVALID ASSERTED; AWVALID = 1
AWREADY from DUT; AWREADY = 1
Address ; AWADDR = 0000e8c6
AWVALID = 0 ; AWADDR = 00000000
----------------------------------------------------------------
Write data channel
Driving Write data sequence to dut
WVALID ASSERTED; WVALID = 1
WREADY from DUT; WREADY = 1
Data ; WDATA = 00008559
Printing from DUT
 Packetized address and data = 5000085590000e8c6
WVALID = 0 ; WDATA = 00000000
----------------------------------------------------------------
Printing from DUT
 Packetized address and data = 5000085590000e8c6
```

Figure 7. DATA Loading into the DUT Write Transaction

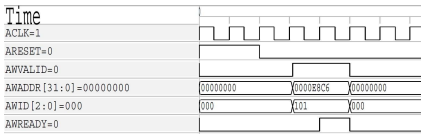


Figure 8. Write Address channel waveform

The waveform in figure 8. depicts the value of the write address from the AXI4 Lite master side being sent over the slave interface. Here the address is "0000E8C6".

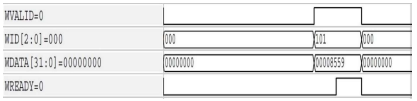


Figure 9. Write Data channel waveform

The waveform in figure 9. depicts the value of the write data from the AXI4 Lite master side being sent over the slave interface. Here the data is "00008559".

```
WRite transaction completed
----------------------------------------
Read address channel
driving Read Address sequence
ARVALID asserted; ARVALID = 1
ARReady asserted; ARREADY = 1
Address; ARADDR = 0000e8c6
ARVALID = 0 ; ARADDR = 00000000
----------------------------------------
Read data channel
printing from DUT
 RVALID is asserted; RDATA is Valid
 Printing from testbench
RREADY asserted; RDATA = 00008559
scoreboard comparing the data
The data received is correct
----------------------------------------
```

Figure 10. DATA Loading into the DUT Read Transaction

Figure 10. demonstrates how data is loaded in DUT for a read transaction. Both ARVALID and ARREADY are 1 for read address channel. The scoreboard data matches the actual data here. The RDATA value is "00008559".

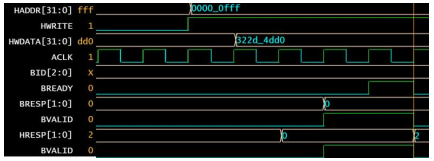


Figure 11. BRESP and HRESP generation

Here the testbench is driving the AHB bridge. The master AHB writes the data to the memory, generating HRESP signal. This will also generate a BRESP signal to AXI4 Lite. Hence, address and data information is obtained in the slave memory. This completes transaction.

Summary/Conclusions

In this project, a simple UVM based verification testbench for AXI4-lite to AHB bridge is developed. The developed test environment can be reused with new functions for different specifications. The testbench c checks the functional correctness of the bus bridge, identify issues in handshake in the AXI bus. The UVM testbench is reusable for other tests that can be created by using the same hierarchy. The testbench is run on Synopsys VCS, UVM 1.2 library, with GTKwave for simulation waveform and debugging.

Key References

[1] "AXI Reference Guide - Xilinx." 21 Sep. 2010, https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf. Accessed 23 Apr. 2020.

[2] "ARM AMBA 5 AHB Protocol Specification AHB5, AHB-Lite." 6 Jun. 2006, https://static.docs.arm.com/ihi0033/bb/IHI0033B_B_amba_5_ahb_protocol_spec.pdf. Accessed 23 Apr. 2020.

[3] "AHB-Lite to AXI4 Bridge v3.0 LogiCORE IP Product ... - Xilinx." 18 Nov. 2015, https://www.xilinx.com/support/documentation/ip_documentation/ahblite_axi_bridge/v3_0/pg176-ahblite-axi-bridge.pdf. Accessed 23 Apr. 2020.

[4] "Verification Methodology Cookbooks | Coverage, UVM and OVM." https://verificationacademy.com/cookbook. Accessed 1 May. 2020.

Acknowledgements

We would like to thank our project advisor Prof. Dr. Lili He for her support and co-operation in this project. We would also like to thank our Co-advisor Prof. Morris Jones for his contribution in selecting the project and guiding us towards achieving the overall project goal by the best possible approach.

We also appreciate San Jose State University for providing us access to software resources that helped us in the successful completion of the project.