ECE 593: FUNDAMENTALS OF PRE-SILICON VALIDATION (SPRING 2021)

Title : AMBA 3 AXI VERIFICATION REPORT

Guide: Tom Schubert

Team Members:

Erik Fox(erfox@pdx.edu)

Keerthi Chowdary Yerragangu(key2@pdx.edu)

Abstract:

We verified AMBA 3 AXI bus communication protocol functionality using testbench components by generating the deterministic and random tests by using a Scoreboard.

Design components:

1) AXI top design - **AXI_top_design.sv**
2) AXI design interface - **AXI_Interface.sv**
3) AXI master- **AXI_master.sv**
4) AXI slave - **AXI_slave.sv**

I.Verification Techniques:

i) Test bench components:

1) Testbench Top- **top.sv**
2) Interconnect between design and testbench - **duv.sv**
3) Test bench interface - **tb BFM.sv**
4) Environment - **axiprotocol_pkg.sv**
    1) **Class request** - has constraints
    2) **Class tester** - generator
    3) **Class coverage** -collects coverage
    4) **Class scoreboard** - checker
    5) **Class testbench** - calls coverage,scoreboard,tester components

ii) Testbench Interactions:

The top module in top.sv has testbench interface instance tbbfm bfm(),duv instance [d0] and Calls class testbench task execute which inturn calls tester, coverage, scoreboard.

- The function of a class tester is to generate inputs both deterministic and constrained random stimulus based on class request.
    01. The function of class request is to constraint the inputs based on the protocol .
- The class coverage collects coverage of inputs, outputs . As we did the grey box model we have collected coverage for transition of internal states in master and slave .

- The class scoreboard is a checker which checks multiple read bursts,overlapping reads,out of order transactions; in case of write it checks for multiple bursts, out of order processing.

The design is connected to the testbench via duv.sv by tbBFM.sv[which is the testbench interface].

II.VERIFICATION TESTS , RESULTS AND COVERAGE :-

A)We have done some **deterministic tests** to check the protocol core functions like read burst, write burst; additional functionality like fixed length read bursts,incrementing readbursts,wrapping read burst,out of order transactions.

1. In detail- fixed length read burst with size 1,2,4 ;incrementing read burst with read size 1,2,4; wrapping read burst with read length 2 and read size 1,2,4; wrapping readburst with read length 4 and read size 1,2,4; wrapping read burst with read length 8 and read size 1,2,4; wrapping readburst with read length 16 and read size 1,2,4.
   To check above functionality we have sequentially written from 32'h00000600 to 32'h0000060F and did read bursts of all types fixed,incrementing,wrapping of different lengths and read sizes.
2. Some other cases checked are writing, reading from the same location,simultaneous writes to different memory locations,back to back writes to same memory location and reading atlast,accessing out of bound memory.

B) We have done **static constrained randomization** to select between different operations of single read, single write, overlapping readbursts, transaction ordering.

C)Test bench is enhanced to **accept $plusargs parameter** to select between different tasks like reset , deterministic tests, read burst,write burst,overlapping read,out of order transactions.

D)Done **Constrained randomization** according to axi3 specification constraints kept in repeat loop of 1000

E)Used **golden vector model of checking** - we are able to confirm if a read supports-multiple bursts,overlapping reads,out of order processing and for writes-multiple bursts,out of order processing.

F)Regarding **code coverage**, we are able to get 75% of overall code coverage with AXI_master ,AXI_slave coverages of 97%,88% respectively. We are able to improve coverage of both master and slave by running for more simulation time, especially the slave.This is because

running more tests and for more time all the different cases in slave got hit-so improved coverage.

Regarding **functional coverage**,we wrote covergroups for all input , output signals and internal state transitions for both master and slave-write address ,write data,write response,read address, read data . We observed that there is 91% coverage in state transitions of master and 98% state transition coverages in slave. As we are constraining inputs, we got input coverage of 71%. The output coverage is 87% - which includes cross coverages also.We got less cross coverage for writeresponse cross which is 45%.

III.EFFORT:

During starting weeks, emphasis was on understanding the protocol later finding out understandable design to verify.Discussed and collaborated on framing Verification plan -like going for grey box type testing, using golden vector model for checking, what functional coverages to cover - capturing internal state transitions etc,. Later we have written a generalised testbench with basic deterministic inputs to make sure the design is working and we are able to see waveform. At this point we discussed how to give different inputs like overlapping, out of order inputs .As we made sure we are able to drive inputs and code is working. We discussed and came up with a testbench framework i.e; like a bus functional model to have a separate interface for testbench, a DUV to connect our testbench with design , a package which consists of generator,coverage,constraints,scoreboard components.Implementing the above said framework in oops concepts. Learning,application of oops concepts in testbench , interconnecting between different modules .Tried out different ways to improve coverages,debugging whenever required. Updated the Verification plan based on what we have done so far and things not covered. Worked on a project report.

Our work - erik-fox/AXIProtocol-Verification (github.com).

IV.CHALLENGES ENCOUNTERED:

1. Learning and implementing testbench in oops concepts is a challenge as it is first time for both of us.
2. Writing hierarchical testbench, interconnecting between various modules ,and different classes took a lot of time.
3. Writing a scoreboard and making sure it works and being able to detect errors is a daunting task.

V. CODE LEVERAGED:

Source code is taken from GITHUB [yvnr4you/AMBA_AXI3: System Verilog and Emulation. Written all the five channels. (github.com)]

VI.BUGS

We are able to detect errors with our scoreboard. Below is the analysis of errors from the scoreboard.

1428 Errors in 30,000 cycles
-Note Multiple errors possible per cycle

133 Invalid RLast signal
199 Invalid WLast signal
3 Master signal not matching input on ARREADY
0 Master signal not matching input on ARVALID
0 Master signal not matching input on AWREADY
0 Master signal not matching input on AWVALID
247 Master signal not matching input on address change
183 Master not applying valid on address change, necessary for overlapping readbursts
3 No matches in the queue for signal changes
106 Missing WLAST signal from slave on last burst
90 Premature WLAST
309 Premature RLast
156 Missing RLAST signal from slave on last burst
0 BID doesn't match WID

VII.FUTURE PLANS:

As of now we verified for multiple read bursts,overlapping reads,out of order transactions; in case of write it checks for multiple bursts, out of order processing. If we have more time we would have verified additional features like system cache support,protection unit support ,atomic operations.