Sumeet Pawar
Surakshith Reddy Mothe
Seth Rohrbach
ECE593

AXI4-Lite Verification Plan

I. Verification Requirements

A. Verification Levels

The AXI4-Lite protocol will be divided into two verification levels:
i) Unit Level
ii) System Level
We chose these verification levels due to the complexity of individual components involved.

i) Unit Level:
The AXI4-Lite protocol consists of two units:

a) Master
b) Slave

The Master and Slave are both implemented as a finite state machine (FSM). Since these both units have their own set of functionality and complexity, we decided to verify them individually before moving on to the System level. Also, at this level, we have better controllability as well as observability to test the units.

a) Master:
For verifying the Master unit, we can have two generators. One generator would be an initiator and the other generator would be a responder while our Master unit would be acting as the design under test (DUT). Here, the CPU will be acting as an initiator and the Slave unit will be acting as a responder. The CPU (initiator) will provide its requests to the Master DUT, upon receiving these requests from the CPU, the Master DUT will issue appropriate signals to the Slave. The Slave (responder) will provide appropriate stimulus to the Master DUT upon observing the signals generated by the Master DUT.

b) Slave:
For verifying the Slave unit, we can have only one generator which will act as an initiator. There is no need for a responder because the Memory is itself implemented in the slave module in the given design. If this was not the case, we would have been required to implement the Memory as a responder. Here, the Master unit will be acting as an initiator while our Slave unit will act as a DUT. The Master (initiator) will provide signals to the Slave DUT and the Slave DUT will access the Memory and respond with the appropriate signals to the Master.

Observing and checking the signal activity on the input and output of Master DUT and Slave DUT will help us verify these units by discovering and fixing any bugs. These bugs may be present in the DUT or even in our verification environment.

ii) System Level:
After completing the unit level testing, we will perform System Level testing for the AXI4-Lite protocol. At this level, we will be assuming that both units, Master and Slave, have been verified and are functionally correct. The main focus at this level would be on interfacing the Master and Slave units together. After integrating both these units together, we will perform overall testing of our AXI4-Lite protocol.

Here, we would require one generator which will be acting as an initiator. The CPU will be acting as an initiator while our AXI4-Lite Bus will be acting as a DUT.
The CPU (initiator) will generate CPU requests which will be provided to the AXI4-Lite Bus. Our AXI4-Lite design will respond to those CPU requests. Observing and checking the signals through which we have integrated the Master and Slave units together will help us verify our entire design at System Level. We expect to find fewer bugs at System Level as compared to the Unit Level.

B. Functions

i. Critical Functions
AXI4-Lite Read Transaction:

- ➢ The Master puts an address on the Read Address channel while asserting ARVALID to indicate the address is valid, and RREADY to indicate the master is ready to receive data from the slave.
- ➢ The Slave asserts ARREADY to indicate that it is ready to receive the address on the bus.
- ➢ Since both ARVALID and ARREADY are asserted, on the next rising clock edge the handshake occurs, after this the master and slave deassert ARVALID and the ARREADY, respectively. Note that, at this point, the slave has received the requested address.
- ➢ The Slave puts the requested data on the Read Data channel and asserts RVALID, indicating the data in the channel is valid. Since both RREADY and RVALID are asserted, the next rising clock edge completes the transaction. RREADY and RVALID can now be deasserted.

AXI4-Lite Write Transaction:

- ➢ The Master puts an address on the Write Address channel and data on the Write data channel while asserting AWVALID and WVALID to indicate the address and data on the respective channels is valid. The Master also asserts BREADY to indicate it is ready to receive a response.
- ➢ The Slave asserts AWREADY and WREADY on the Write Address and Write Data channels, respectively to indicate it is ready for a write transaction.
- ➢ Since Valid and Ready signals are present on both the Write Address and Write Data channels, the handshakes on those channels occur and the associated Valid and Ready signals can be de-asserted. Note that, at this point, the slave has the write address and data.
- ➢ The Slave asserts BVALID, indicating there is a valid response on the Write response channel. The next rising clock edge completes the transaction, with both the Ready and Valid signals on the write response channel high.

Reset Logic: ARESETN is an active-low asynchronous reset which sets all the handshake signals VALID/READY in all channels to low.

ii. Secondary Function:

There are no secondary functions as the design is a communication protocol and can act as a block for another system.

iii. Non-Verified Function:

There are no non-verified functions as the design is a communication protocol and can act as a block for another system.

C. Specific Tests & Methods

i. Type of Verification

We will test the AXI4-Lite protocol in a grey box environment. Since it is a communication protocol, the internal functioning of the device is of secondary importance to the black box validity of the design; however, since we are starting with a stable design, the possibility of accessing the internals may be of value for ease of debugging and validation.

ii. Verification Strategy

~~We will drive the device by writing packets of data through the AXI4-Lite bus and writing the results to a local memory. We then confirm validity by reading them back through the bus and checking the results with the original value.~~

Our verification strategy is to use a combination of deterministic, constrained random, and fully random tests to maximize coverage of the design. The deterministic tests will focus on a subset of the address space as well as some of the corner cases. The constrained random tests will target a greater range of potentially problematic areas of the design. The fully randomized tests will be used to improve coverage of the design.

We will implement an object-oriented constrained random test bench. This should enable us to get full coverage of the design.

iii. Abstraction Level

~~The AXI4-Lite is a communication device, so we will be verifying it at a packet level. Driving entire packets into the device and ensuring the output is correct will most closely emulate the device's actual purpose.~~

We are using a reference model checker strategy. Effectively, our test bench replicates the function of the master module and acts as both a master module and CPU during the tests. It also receives a copy of all values read back from the slave module in order to confirm the accuracy of read operations.

iv. Checking

We perform checking by comparing the data written to memory via the AXI4-Lite bus with the value sent. When there is a mismatch, it is flagged and a message is printed to the terminal. We will also implement a scoreboard to track the success rate.

D. Coverage

We will be gathering both code coverage and functional coverage details. Code coverage will help us identify unexercised portions of the design which will in turn help us to include appropriate test cases in our verification environment in order to trigger the unexercised part of code. Functional coverage will help us identify if any functionality is missing from the design. Both the code and functional coverage details will help us track our verification progress.

Coverage goals:
i) The stimulus has created a specific or varying range of signals.
ii) The environment has exercised all types of commands and transactions.
iii) The environment has driven varying degrees of legal concurrent stimulus.
iv) The initiator and responder components have driven errors into the DUV.

We would be forming multiple covergroups which will comprise of the signals that we want to be sampled together. Inside these covergroups, we would be having appropriate coverpoints and bins.

Covergroup details:
We can have the following 9 covergroups (6 for Code coverage and 3 for Functional coverage) in order to obtain the code and functional coverage data:

I) Code Coverage:
i) Covergroup for Read Address channel:
Here, inside this cover group, we would be maintaining different coverpoints for the following signals:

a) Coverpoint for ARADDR_STABLE_Test
b) Coverpoint for ARADDR_X_Test
c) Coverpoint for ARVALID_STABLE_Test
d) Coverpoint for ARREADY_STABLE_Test
e) Coverpoint for ARVALID_X_Test
f) Coverpoint for ARREADY_X_Test
g) Coverpoint for READ_ADDRESS_Test

ii) Covergroup for Read Data channel:
Here, inside this cover group, we would be maintaining different coverpoints for the following signals:

a) Coverpoint for READ_DATA_Test
b) Coverpoint for RDATA_X_Test
c) Coverpoint for RDATA_STABLE_Test
d) Coverpoint for RVALID_STABLE_Test
e) Coverpoint for RVALID_X_Test

f) Coverpoint for RREADY_STABLE_Test
g) Coverpoint for RREADY_X_Test

iii) Covergroup for Write address channel:
Here, inside this cover group, we would be maintaining different coverpoints for the following signals:

a) Coverpoint for AWADDR_STABLE_Test
b) Coverpoint for AWADDR_X_Test
c) Coverpoint for AWVALID_STABLE_Test
d) Coverpoint for AWVALID_X_Test
e) Coverpoint for AWREADY_STABLE_Test
f) Coverpoint for AWREADY_X_Test
g) Coverpoint for WRITEADDRESS_Test

iv) Covergroup for Write data channel:
Here, inside this cover group, we would be maintaining different coverpoints for the following signals:

a) Coverpoint for WVALID_STABLE_Test
b) Coverpoint for WVALID_X_Test
c) Coverpoint for WREADY_STABLE_Test
d) Coverpoint for WREADY_X_Test
e) Coverpoint for WDATA_STABLE_Test
f) Coverpoint for WDATA_X_Test
g) Coverpoint for WDATA_Test

v) Covergroup for Write response channel:
Here, inside this cover group, we would be maintaining different coverpoints for the following signals:

a) Coverpoint for BVALID_STABLE_Test
b) Coverpoint for BVALID_X_Test
c) Coverpoint for BREADY_STABLE_Test
d) Coverpoint for BREADY_X_Test

vi) Covergroup for CPU signals:
Here, inside this cover group, we would be maintaining different coverpoints for the following signals:

a) Coverpoint for rd_en
b) Coverpoint for wr_en
c) Coverpoint for Read_Address
d) Coverpoint for Write_Address
e) Coverpoint for Write_Data

II) <u>Functional Coverage:</u>
i) Covergroup for Master FSM:
Here, inside this cover group, we would be maintaining different coverpoints for the FSM transitions:

a) Coverpoint for MASTER READ:

Here we will have transition bins for gathering the coverage data for MASTER READ FSM transitions.

b) Coverpoint for MASTER WRITE:
Here we will have transition bins for gathering the coverage data for MASTER WRITE FSM transitions.

ii) Covergroup for Slave FSM:
Here, inside this cover group, we would be maintaining different coverpoints for the FSM transitions:

a) Coverpoint for SLAVE READ:
Here we will have transition bins for gathering the coverage data for SLAVE READ FSM transitions.

b) Coverpoint for SLAVE WRITE:
Here we will have transition bins for gathering the coverage data for SLAVE WRITE FSM transitions.

iii) Covergroup for Reset signal:
This covergroup will be triggered only when ARESETN signal is asserted. Here, inside this cover group, we would be maintaining different coverpoints for the following signals:

a) Coverpoint for RREADY
b) Coverpoint for ARVALID
c) Coverpoint for AWVALID
d) Coverpoint for WVALID
e) Coverpoint for BREADY
f) Coverpoint for ARREADY
g) Coverpoint for RVALID
h) Coverpoint for AWREADY
i) Coverpoint for WREADY
j) Coverpoint for BVALID

This covergroup will let us know if the coverpoint signals were de-asserted upon the ARESETN signal going low.

E. Scenarios

Unit Level Tests:

Read Address Channel:

Test Label	**Description**
ARVALID_Reset_Test	ARVALID should be low when ARESETN is low

ARREADY_Reset_Test	ARREADY should be low when ARESETN is low
ARADDR_Valid_Test	ARADDR value should be valid and stable when ARVALID is high
ARVALID_Stable_Test	When ARVALID is asserted high, then it must remain high until ARREADY is high
ARREADY_Stable_Test	ARREADY is asserted, then it remains asserted until ARVALID is high
ARVALID_Valid_Test	ARVALID value should be valid when not in reset i.e, ARESETN is high
ARREADY_Valid_Test	ARREADY value should be valid when not in reset i.e, ARESETN is high

Read Data Channel:

RVALID_Reset_Test	RVALID should be low when ARESETN is low
RREADY_Reset_Test	RREADY should be low when ARESETN is low
RDATA_Valid_Test	RDATA value should be valid and stable when RVALID is high
RVALID_Stable_Test	When RVALID is asserted high, then it must remain high until RREADY is high
RVALID_Valid_Test	RVALID value should be valid when not in reset i.e, ARESETN is high
RREADY_Stable_Test	When RREADY is asserted high , then it must remain high until RVALID is high
RREADY_Valid_Test	RREADY value should be valid when not in reset i.e, ARESETN is high

Write Address Channel:

AWVALID_Reset_Test	AWVALID should be low when ARESETN is low
AWREADY_Reset_Test	AWREADY should be low when ARESETN is low
AWADDR_Valid_Test	AWADDR value should be valid and stable when AWVALID is high

AWVALID_Stable_Test	When AWVALID is asserted high, then it must remain high until AWREADY is high
AWVALID_Valid_Test	AWVALID should be valid when not in reset i.e, ARESETN is high
AWREADY_Stable_Test	When AWREADY is asserted high, then it must remain high until AWVALID is high
AWREADY_Valid_Test	AWREADY should be valid when not in reset i.e, ARESETN is high

Write Data Channel:

WVALID_Reset_Test	WVALID should be low when ARESETN is low
WREADY_Reset_Test	WREADY should be low when ARESETN is low
WVALID_Stable_Test	When WVALID is asserted high, then it must remain high until WREADY is high
WVALID_Valid_Test	WVALID should be valid when not in reset i.e, ARESETN is high
WREADY_Stable_Test	When WREADY is asserted high, then it must remain high until WVALID is high
WREADY_Valid_Test	WREADY value should be valid when not in reset i.e, ARESETN is high
WDATA_Valid_Test	WDATA should be valid and stable when WVALID is high

Write Response Channel:

BVALID_Reset_Test	BVALID should be low when ARESETN is low
BREADY_Reset_Test	BREADY should be low when ARESETN is low
BVALID_Stable_Test	When BVALID is asserted high, then it must remain high until BREADY is high
BVALID_Valid_Test	BVALID value should be valid when not in reset i.e, ARESETN is high

BREADY_Stable_Test	When BREADY is asserted high, then it must remain high until BVALID is high
BREADY_Valid_Test	BREADY value should be valid when not in reset i.e, ARESETN is high

System Level Tests:

Test Label	**Description**
ARESETN_Test	ARESETN is an active-low asynchronous reset which sets all the handshake signals VALID/READY in all channels to low
Write_Read_IntermediateLocation_Test	Perform a single write and then single read from the same intermediate location to see the data matched
Write_Read_FirstLocation_Test	Perform a single write and then single read from the first location to see the data matched
Write_Read_LastLocation_Test	Perform a single write and then single read from the last location to see the datas matched
Write_Read_Sametime_SameLocation_Test	Perform single read and single write at the same time to the same location (including the first and last locations) will cause the read to happen first and then write operation will take place and data is written to memory.
Write_Read_Sametime_DifferentLocation_Test	Perform single read and single write at the same time to different locations and check proper data is written and read to respective location at same time
Multiple_Writes_Multiple_Reads_ConsecutiveLocations_Test	Perform multiple writes first to consecutive locations and then multiple reads from the same consecutive locations to see the data matched for each location
Multiple_Writes_Multiple_Reads_RandomLocations_Test	Perform multiple writes first to random locations and then multiple reads from the same random

	locations to see the data matched for each location
Multiple_Writes_Single_Read_SameLocation_Te st	Perform multiple writes to same location and finally read from the same location to see the last written data matched
Sudden_Reset	Set ARESETN low and check that it does not affect the contents of the memory
Outofboundary_Memory_Access_Test	Try accessing a memory location which is off the limits of the available memory

II. Project Management

A. Tools

For this project, we will be using QuestaSim for all simulation needs. All members of the team are students at Portland State University and have access to the necessary software through remote access of the school's Linux servers.
For collaboration, we are making use of a combination of text messaging, Zoom meetings, and Google suite tools.

B. Risks & Dependencies

The AXI4-Lite is an industry-standard design, complete with the specification, so changing the design is a minimal risk for this project.
All members of the team are familiar with the tools we expect to be necessary to complete the project, so there should be minimal risk of problems with learning new tools.
The team is expected to be static for the duration of the project. The schedule is a relatively short one, so there is minimal risk of unexpected team changes in the coming weeks.
Perhaps the biggest risk for successful completion of the project is complacency. If all team members remain vigilant and on-schedule with individual deliverables, there should be minimal risk overall.
The RTL source code that the project is dependent on has been secured by the team already. We also have the specification for the protocol.
We are dependent on the university's Linux servers for all simulation. Unfortunately, licenses to industrial grade simulation software are quite expensive so purchasing individual licenses as a fallback is not feasible. However, the servers are in general extremely reliable so the risk of losing all access to them is minor.

C. Resources

The most important resources for this project are the 3 team members: Sumeet Pawar, Surakshith Reddy Mothe, and Seth Rohrbach. In addition, QuestaSim through Portland State University's Linux servers will be a critical resource. In the event we run into a seemingly insurmountable problem, consulting with Dr. Schubert will be an important resource.

D. Schedule

May 11 - Submit Verification Plan

- Finish planning stage and submit plan for review.

May 15 - Submit Project Waveform

- Produce and submit a waveform which demonstrates that the RTL is functional and we can drive the majority of the outputs.

May 29 - Submit Initial Code

- Submit the code we are using at this point, along with the test bench.

June 5 - Submit Final Code and Report

- End of project. Submit all deliverables.