Final Project Verification Draft

AXI4-Lite protocol

ECE 571 - SystemVerilog

Winter 2020

The Advanced eXtensible Interface (AXI),

a part of the ARM Advanced Microcontroller

Bus Architecture

Team 8:

Disha Shetty

Likitha Atluru

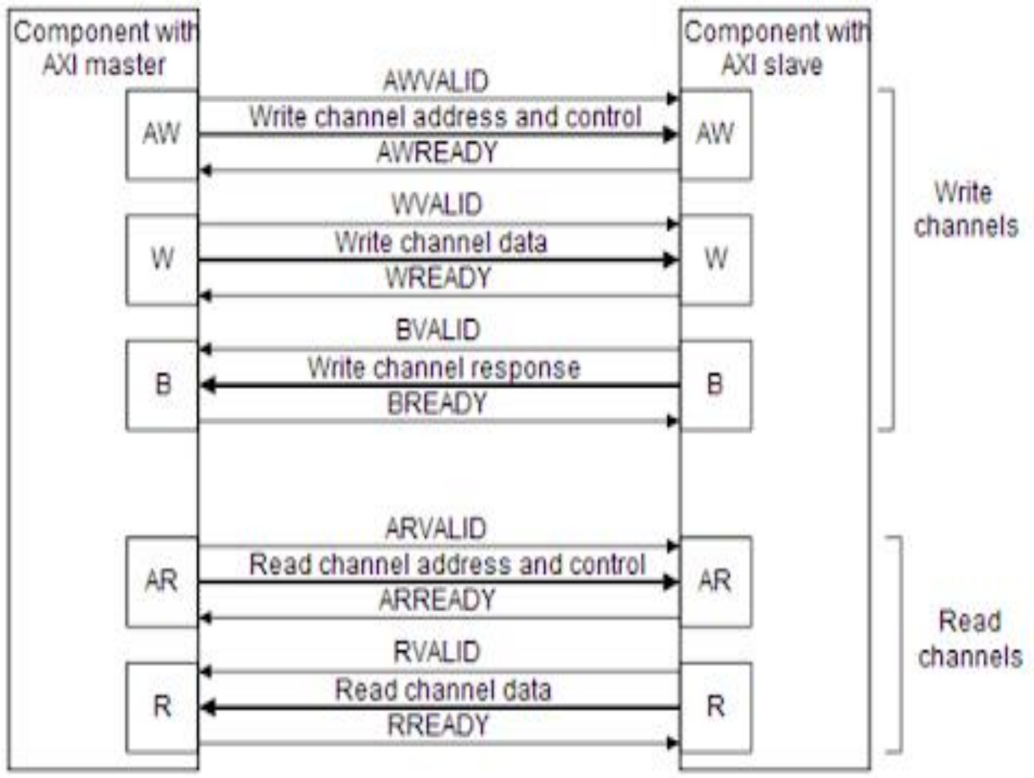
Preetha Selvaraju

Rutuja Patil

Introduction:

An important aspect of an SoC is not only which components or blocks it houses, but also how they interconnect. AMBA is a solution for the blocks to interface with each other. The ARM **Advanced Microcontroller Bus Architecture** (**AMBA**) is an open-standard, on-chip interconnect specification for the connection and management of functional blocks in system-on-a-chip (SoC) designs. The AMBA AXI-Lite protocol is used for communication with simpler control register style interfaces within components.

With increasing numbers of CPU cores, subsystems and communication IPs in today's System-on-Chips, verification is the biggest challenge in the design of (SoC) devices. Therefore, as a final project for the course ECE571 - Introduction to SystemVerilog for Design and Verification at Portland State University, the team will design the entire system from scratch and execute a good verification strategy for **AXI4-Lite protocol** using SystemVerilog verification techniques such as checkers, assertions, randomization, etc.



Design Approach:

We are designing the AXI-4 Lite Protocol from scratch and following is our design approach :

AXI-4 Lite Protocol design includes separate SystemVerilog files for package definition, interface, and modules.

Package definitions: We have defined the parameters including Address Width and Data Width . We are considering address width as 64 bits.

Interface: Interfaces are used for communication between master and slave module. In the AXI-4 lite interface we have defined all the signals for various channels including Read Address Channel, Write Address Channel, Read Data Channel, Write Data Channel and Write Response Channel. Interface also includes modport for master and slave to define its input and output ports.

Master and Slave module:

Master module:

AXI-4 Lite is used for unidirectional transaction. Master module is designed using Finite State Machine, we have two different FSM to read and write data. Master will send signals to slave to read data from memory or to write data to memory.

Slave module:

Slave module is designed using two Finite State Machine in which one describes functionality for read transaction and other for write transaction.

Read Transaction:

1. Master puts address on read address channel as well as assert ARVALID to indicate that address is valid and assert RREADY to indicate master is ready to accept data from slave.

2. After slave indicates its ready to accept address handshake process will start and slave will place data on read data channel and assert RVALID to indicate data is valid.

3. Since both RREADY and RVALID is asserted the transaction will be completed in next cycle and both the signals will be deasserted.

Write Transaction

1. Master will put address on Write Address Channel and data on Write Data channel and will indicate address and data are valid by asserting AWVALID and WVALID respectively. Master also assert BREDAY to indicates its ready to receive response.

2. Slave will assert AWREADY and WREADY on Write Address and Write data Channel respectively,

3. Since we have Ready signals on both Write Address and Write Data channel handshake will occur and ready signals can be deasserted.

4. Slave will then assert BVALID to indicate there is valid response on Write Response Channel and on next rising edge of clock transaction will complete.

Verification Approach:

- We will familiarize ourselves with the Advanced Microcontroller Bus Architecture (AMBA) and focus on the modules that are compatible with the AXI4-Lite protocol, which provides a simpler register style interface to the systems. A set of signals in each channel are used to define a set of unit tests. Protocol functionality specifies the timing relationship between signals.
- Mentor Graphics Questa Simulator is used for SystemVerilog simulation and there by expanding Verification capabilities to Assertion based verification.
- Along with the unit tests, a set of assertions for each channel will be defined in a separate file and bound to the design. Each independent channel will have its own separate testbench. For all these channels, there will be assertions that run along each testbench. Also, a Makefile will be used for compiling and running each testbench.
- Finally, this assertion-based verification environment will be used to verify our design implemented from scratch, in order to prove its effectiveness and implementation to find bugs.

Verification plan:

- Unit level and system level design verification will be implemented.
- After reviewing and designing the AXI4-lite protocol functionality, we proceeded to consider each signal in different channels and implement different possibilities to define the following basic unit tests.

Unit Tests:

- For unit level testing, we will be writing directed test cases to check the functionality of each channel.
- Assertions will be used as checkers for unit level testing. A brief test plan for each unit is provided below.

Read Address testbench:

Following are the signals in this channel:

ARADDR

ARVALID

ARREADY

When ARVALID and ARREADY are asserted, it is tested for successful transaction of address (ARADDR) from master to slave.

Assertions:

Property name	**Channel**	**Description**	**Owner**
ARADDR_STABLE_a	Read address	ARADDR remains stable when ARVALID is asserted and ARREADY is low	Likitha
ARADDR_X_a	Read address	A value of ‘X’ on ARADDR is not permitted when ARVALID is HIGH.	Likitha
ARVALID_RESET_a	Read address	ARVALID is LOW for the first cycle after ARESETn goes HIGH	Likitha
ARREADY_RESET_a	Read address	When ARESETN goes LOW ARREADY should be LOW	Likhita
ARVALID_STABLE_a	Read address	When ARVALID is asserted, then it remains asserted until ARREADY is HIGH	Likitha

ARREADY_STABLE_a	Read address	When ARVALID is asserted, then it must remain asserted until ARREADY is HIGH	Likitha
ARVALID_X_a	Read address	A value of ‘X’ on ARVALID is not permitted when not in reset.	Likitha
ARREADY_X_a	Read address	A value of ‘X’ on ARREADY is not permitted when not in reset.	Likitha
READADDRESS_a	Read address	Check whether ARADDR has been received successfully by the slave	Likitha

<u>Read</u>

<u>Data testbench</u>:

Following are the signals in this channel:

RDATA

RVALID

RREADY

If RREADY and RVALID are asserted, it is tested whether RDATA is being transmitted from slave to master.

Assertions:

Property name	Channel	Description	Owner
RDATA_STABLE_a	Read Data	RDATA remains stable when RVALID is asserted, and RREADY is LOW.	Preetha
RDATA_X_a	Read Data	A value of X on RDATA valid byte lanes is not permitted when RVALID is HIGH.	Preetha
RVALID_RESET_a	Read Data	RVALID is LOW for the first cycle after ARESETn goes HIGH	Preetha
RREADY_RESET_a	Read Data	When ARESETN goes low RREADY should be LOW	Preetha
RVALID_STABLE_a	Read Data	When RVALID is asserted, then it must remain asserted until RREADY is HIGH	Preetha

RREADY_STABLE_a	Read Data	When RREADY is asserted, then it must remain asserted until RVALID is HIGH	Preetha
RVALID_X_a	Read Data	A value of X on RVALID is not permitted when not in reset	Preetha
RREADY_X_a	Read Data	A value of X on RREADY is not permitted when not in reset	Preetha

Write Address Testbench:

Following are the signals in this channel:

AWADDR

AWVALID

AWREADY

When AWVALID and AWREADY are asserted, check points are asserted to test successful transaction of address (AWDDR) from master to slave.

Assertions:

Property name	**Channel**	**Description**	**Owner**
AWADDR_STABLE_a	Write Address	AWADDR remains stable when AWVALID is asserted and AWREADY is LOW	Disha

AWADDR_X_a	Write Address	A value of X on AWADDR is not permitted when AWVALID is HIGH	Disha
AWVALID_RESET_a	Write Address	AWVALID is LOW for the first cycle after ARESETn goes HIGH	Disha
AWREADY_RESET_a	Write Address	When ARESETN goes LOW AWREADY should be LOW	Disha
AWVALID_STABLE_a	Write Address	When AWVALID is asserted, then it remains asserted until AWREADY is HIGH	Disha
AWREADY_STABLE_a	Write Address	When AWREADY is asserted, then it must remain asserted until AWVALID is HIGH	Disha
AWVALID_X_a	Write Address	A value of X on AWVALID is not permitted when not in reset	Disha

AWREADY_X_a	Write Address	A value of X on AWREADY is not permitted when not in reset	Disha
WRITEADDRESS_a	Write Address	Check whether AWADDR has been received successfully by the slave	Disha

Write Data testbench:

Following are the signals in this channel:

WDATA

WVALID

WREADY

If WREADY and WVALID are asserted, it is tested whether WDATA is being transmitted from master to slave. Test cases which check if transaction is completed after the WREADY and WVALID being de-asserted is implemented.

Assertions:

Property name	**Channel**	**Description**	**Owner**
WDATA_STABLE_a	Write Data	A value on WDATA remains stable when WVALID is asserted and WREADY is LOW.	Rutuja

WDATA_a	Write Data	Check whether data Write data has been received successfully by the slave	Rutuja
WDATA_X_a	Write Data	A value of X on WDATA valid byte lanes is not permitted when WVALID is HIGH.	Rutuja
WVALID_RESET_a	Write Data	WVALID should be LOW for the first cycle after ARESETn goes HIGH.	Rutuja
WREADY_RESET_a	Write Data	When ARESETN goes low WREADY should be low	Rutuja
WVALID_STABLE_a	Write Data	When WVALID is asserted, then it must remain asserted until WREADY is HIGH.	Rutuja
WREADY_STABLE_a	Write Data	When WREADY is asserted, then it must remain asserted until WVALID is HIGH	Rutuja

WVALID_X_a	Write Data	A value of X on WVALID is not permitted when not in reset.	Rutuja
WREADY_X_a	Write Data	A value of X on WREADY is not permitted when not in reset.	Rutuja

Write Response Testench:

Following are the signals in this channel:

BRESP

BVALID

BREADY

If BVALID and BREADY are asserted, it is tested whether BRESP is ‘OKAY’ indicating the successful write operation.

Assertions:

Property name	Channel	Description	Owner
BREADY_STABLE_a	Write Response	When BREADY is asserted, then it must remain asserted until BVALID is HIGH	Disha

BREADY_RESET_a	Write Response	When ARESETN goes low BREADY should be LOW	Disha
BVALID_RESET_a	Write Response	BVALID is LOW for the first cycle after ARESETn goes HIGH	Disha
BVALID_STABLE_a	Write Response	When BVALID is asserted, then it must remain asserted until BREADY is HIGH	Disha
BVALID_X_a	Write Response	A value of X on BVALID is not permitted when not in reset	Disha
BREADY_X_a	Write Response	A value of X on BREADY is not permitted when not in reset	Disha

We have separate testbenches for separate channels and we have transcripts for each of them which I have attached below:

read_address_channel_transcript.txt

read_data_channel_transcript.txt

write_address_channel_transcript.txt

write_data_channel_transcript.txt

write_response_channel_transcript.txt

The top file containing all the assertions as well as some extra assertions for fsm:

Top_with_assertions_transcript.txt

System Level Testbench:

- Verification strategy is that various combinations of write and read transactions are performed and then both the write and read data are verified if they match or not.
- Unit level assertions will be binded for top system level verification.
- The following scenarios are used to verify the functionality of the bus:

1. Default memory value check
   - Check default memory values. (before writing any locations, do read operation we should get default values as ‘h00)

2. Write and Read to a particular memory location
   - Perform write to any memory location, read from the same memory location, read data should be the same as written data.

3. Write and Read to all memory locations
   - Perform write and read to all the memory locations and check if the write and read data matches in all the locations.

4. Reset in Middle of Write/Read Operation
   - Assert reset in between write/read operation and check for default values. (after writing to few locations assert the reset and perform read operation, we should get default memory location value ‘h00)

5. Single Write Operation
   - Perform single write operation to a memory location and check if the data is being written into that location is correct.

6. Single Read Operation
   - Perform single read operation to a memory location and check if the data being read from that location is correct.

7. Multiple Writes followed by Single Read to a memory location
   - Perform multiple writes to a location and then perform read. Check whether the data read matches with the last data that was written.

8. Single Write followed by Multiple Reads to a memory location
    - Perform a single write to a particular location and perform multiple reads on the same location and check if the data being read from all the read transactions matches with the data written into that location first.

top module for system level was run and the following is the transcript for it:

Top_transcript.txt

Team members responsibilities:

- Design: Rutuja Patil, Disha Shetty
- Unit Level Testing:

Read Address Channel:	Likitha Atluru
Read Data Channel:	Preetha Selvaraju
Write Address Channel:	Disha Shetty
Write Data Channel:	Rutuja Patil
Write Response Channel:	Disha Shetty

- System Level Testing: Preetha Selvaraju, Likitha Atluru
- Top level Integration: Likitha Atluru
- Makefiles: Rutuja Patil