PROJECT REPORT

Design and Verification of 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

Abstract:

AMBA AXI4-Lite protocol is designed from scratch and implemented using System Verilog HDL. Then the simulation is carried out for the functionality check using QuestaSim EDA tool. A good verification strategy is executed using SystemVerilog verification constructs such as interfaces, assertions etc. Here, the bus is verified for five different channels and finally system level testing is performed.

Design Approach:

AXI-4 Lite Protocol design includes separate SystemVerilog files for package definition, interface, master module, slave module and top module

- Package definitions:
  Address Width and Data Width are parameterized in a package. Address width is considered as 32 bits.
- Interface:
  Interfaces are used for communication between master and slave module. AXI4-lite interface includes signals for various channels: Read Address Channel, Write Address Channel, Read Data Channel, Write Data Channel and Write Response Channel. Interface specification is enlarged by including modports. A subset of interface elements is defined through modports and for each interface element in the subset, a direction is specified.
- Master module:
  Master module is designed using Finite State Machine, one to read and the other to write data. Master will send signals to slave to read data from memory or to write data to memory.
- Slave module:
  Slave module is also designed using two Finite State Machines in which one describes the functionality for read transaction and other for write transaction. Slave also has memory array with 2^12 locations i.e. 4096 locations for testing purpose. Total memory locations we can consider for our design is 2^32 i.e. 4GB
- Read Transaction Phase:
  - Master puts address on read address channel and asserts ARVALID to indicate that address is valid and asserts RREADY to indicate that master is ready to accept data from slave.
  - When slave indicates that it is ready to accept address, handshake process takes place and slave will place data on read data channel and asserts RVALID to indicate data is valid.
  - Since both RREADY and RVALID are asserted, the transaction will be completed in next cycle and both the signals will be de-asserted.
- Write Transaction Phase:
  - 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 asserts BREADY to indicates that it is ready to receive the response.
  - Slave will assert AWREADY and WREADY on Write Address and Write data Channel respectively.
  - Since Ready signals on both Write Address and Write Data channel are asserted, handshake will take place and ready signals can be de-asserted.
  - 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:

- Advanced Microcontroller Bus Architecture (AMBA) is analyzed and focused 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.
- Assertion-based verification environment will be used to verify each channel including Read Address Channel, Write Address Channel, Read Data Channel, Write Data Channel and Write Response Channel.
- 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.
- System level testbench is implemented to verify the functionality of the design.

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:

Assertions name	Channel	Description
ARADDR_STABLE_a	Read Address	When ARVALID is asserted, address ARADDR remains stable until ARVALID and ARREADY become low
ARADDR_X_a	Read Address	A value of ‘X’ on ARADDR is not permitted when ARVALID is HIGH
ARVALID_RESET_a	Read Address	When ARESETN goes low ARVALID should be low
ARREADY_RESET_a	Read Address	When ARESETN goes low ARREADY should be low
ARVALID_STABLE_a	Read Address	When ARVALID is asserted, then it must remain asserted until ARREADY is HIGH

ARREADY_STABLE_a	Read Address	ARREADY is asserted, then it remains asserted until ARVALID is high
ARVALID_X_a	Read Address	A value of 'X' on ARVALID is not permitted when RESET is high(not in reset)
ARREADY_X_a	Read Address	A value of 'X' on ARREADY is not permitted when ARESETN is high(not in reset)
READADDRESS_a	Read Address	Check whether ARADDR has been received successfully by the slave

- **Read Data testbench**: 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:

Assertions name	Channel	Description
READ_DATA_a	Read Data	Check whether data RDATA has been read successfully by the master
RDATA_X_a	Read Data	A value of X on RDATA valid byte lanes is not permitted when RVALID is HIGH
RVALID_RESET_a	Read Data	when ARESETN goes low RVALID should be low
RREADY_RESET_a	Read Data	When ARESETN goes low RREADY should be low
RVALID_STABLE_a	Read Data	When RVALID is asserted, then it must remain asserted until RREADY is HIGH
RREADY_STABLE_a	Read Data	When RREADY is asserted, then it must remain asserted until RVALID is HIGH
RVALID_X_a	Read Data	A value of X on RVALID is not permitted when ARESETN is high(not in reset)
RREADY_X_a	Read Data	A value of X on RREADY is not permitted when ARESETN is high(not in reset)
RDATA_STABLE_a	Read Data	when RVALID is asserted, RDATA remains stable until RVALID and RREADY become low

- **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:

Assertion name	Channel	Description
AWADDR_STABLE_a	Write Address	when AWVALID is asserted, address AWADDR must remain stable until AWVALID and AWREADY become LOW
AWADDR_X_a	Write Address	A value of 'X' on AWADDR is not permitted when AWVALID is HIGH
AWVALID_RESET_a	Write Address	When ARESETN goes LOW AWVALID should be low
AWREADY_RESET_a	Write Address	When ARESETN goes LOW AWREADY should be low
AWVALID_STABLE_a	Write Address	When AWVALID is asserted, then it must remain asserted until AWREADY is HIGH
AWREADY_STABLE_a	Write Address	When AWREADY is asserted, then it must remian asserted until AWVALID is HIGH
AWVALID_X_a	Write Address	A value of 'X' on AWVALID is not permitted when ARESETN is high(not in reset)
AWREADY_X_a	Write Address	A value of 'X' on AWREADY is not permitted when ARESETN is high(not in reset).
WRITEADDRESS_a	Write Address	Check whether AWADDR has been received successfully by the slave

- **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:

Assertion name	Channel	Description
WDATA_STABLE_a	Write Data	When WVALID is asserted, WDATA must remain stable until WVALID and WREADY become low
WDATA_a	Write Data	Check whether data Write data has been received successfully by the slave
WDATA_X_a	Write Data	A value of X on WDATA valid byte lanes is not permitted when WVALID is HIGH
WVALID_RESET_a	Write Data	when ARESETN goes low WVALID should be low
WREADY_RESET_a	Write Data	When ARESETN goes low WREADY should be low
WVALID_STABLE_a	Write Data	When WVALID is asserted, then it must remain asserted until WREADY is HIGH
WREADY_STABLE_a	Write Data	When WREADY is asserted, then it must remain asserted until WVALID is HIGH
WVALID_X_a	Write Data	A value of X on WVALID is not permitted when ARESETN is high (not in reset)
WREADY_X_a	Write Data	A value of X on WREADY is not permitted when ARESETN is high (not in reset)

- **Write Response Testbench:** Following are the signals in this channel:
  BVALID

  BREADY

Assertions:

Assertion name	Channel	Description
BVALID_RESET_a	Write Response	when ARESETN goes low BVALID should be low
BREADY_RESET_a	Write Response	When ARESETN goes low BREADY should be low
BVALID_X_a	Write Response	A value of X on BVALID is not permitted when ARESETN is high (not in reset)

BREADY_X_a	Write Response	A value of X on BREADY is not permitted when ARESETN is high (not in reset).
BVALID_STABLE_a	Write Response	When BVALID is asserted, then it must remain asserted until BREADY is HIGH
BREADY_STABLE_a	Write Response	When BREADY is asserted, then it must remain asserted until BVALID is HIGH

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.
- Assertions are added to check the proper transition of state from one to another in testbench.
- The following scenarios are used to verify the functionality of the bus:

1. Default memory value check
   - ➢ Check default memory values, in first 20 locations (before writing any locations, do read operation we should get default values as ‘hxxxx)

2. Write to single location.
   - ➢ Perform write to a single memory location and checked whether that particular memory location received the value.

3. Read from single location.
   - ➢ Perform read to a single memory location and check if the write and read data matches in that location.

4. Write and read at same time to different location.
   - ➢ Perform read and write at the same time to different locations and check proper data is written and read to respective location at same time.

5. Write and read at the same time to same location and check if read and write data matches.
   - ➢ Writing and reading to a same location at same time will cause the read to happen first and we get all X values then write operation will take place and data is written to memory.
   - ➢
6. Multiple write followed by multiple reads to consecutive locations and check if the write and read data matches
   - ➢ Performed write and read to consecutive multiple locations and checked whether the data written and read matches.

7. Multiple reads from the consecutive locations.
   - ➢ Performed read operation from multiple consecutive locations and checked whether it matches with the data written to that address.

8. Multiple writes followed by multiple reads to random locations and check if the write and read data matches

- Performed multiple write followed by multiple reads to random locations and check if write and read data matches.

9. Multiple writes followed by a single read to the same location and check if the last data written and read data matches
   - Perform multiple writes to a location and then perform read. Check whether the data read matches with the last data that was written.

10. Consecutive writes to the same location
    - Perform consecutive write operation to same location and check if the last data read matches with the last data written.

11. Single write followed by multiple reads to the same location and check if the read data matches the write data all the time
    - Perform single write operation and perform multiple read operations to that location and check whether read data matches write data all time.

12. Reset in middle of write
    - Giving Reset before completion of write operation and checking whether the write operation is getting aborted

13. reset in middle of read
    - Giving Reset before completion of a read operation and checking whether the write operation is getting aborted

14. Check for out of limit input write address
    - We considered memory location of 4096 hence we can provide write address of 12 bits and here we are checking if the address size limit exceeds12 bit then it should not be taken or give error.

15. Check for out of limit input read address
    - We considered memory location of 4096 hence we can provide read address of 12 bits and here we are checking if the address size limit exceeds12 bit then it should not be taken or give error.

16. Write and read to all locations and check if data matches
    - Perform write and read operation to all locations and check if the data matches.

Challenges Encountered:

Faced issue while implementing array for the size 32 bit address width ,and we tried implementing it using 1D array and associative array. We were able to achieve till 2**31 but not beyond that.

Future Scope:

- As a future scope of our project we can do emulation for AXI4 lite bus using Veloche which will help to produce the verification quickly and efficiently.
- We can include WSTROBE signal for Write data channel, which is a 4 bit signal indicating which of the 4 bytes of write data (32 bit) are we going to write.
- Can use coverage tool from QuestaSim to check functional coverage