# ECE 571 Fall 2021 Final Project

# Verification of a AMBA AXI4-Lite protocol model and then find the bugs injected into the design using your verification environment

**This project is worth 100 points. Final project presentations will be conducted live on either Mon, 06-Dec or Tue, 07-Dec . Presentation slots can be reserved via a Doodle calendar. Deliverables are due on Wed, 08-Dec by 10:00 PM. NO LATE SUBMISSIONS WITHOUT PERMISSION.**

**We will be using the GitHub classroom for this assignment. You should submit your assignment before the deadline by pushing your final code and other deliverables to your team's GitHub repository for the assignment.**

**After completing this assignment students should have:**

- ➢ Gained experience w/ unit level and system level design verification
- ➢ Gained experience w/ SystemVerilog verification techniques using checkers, assertions, randomization, etc
- ➢ Gained experience making a technical presentation
- ➢ Gained experience w/ hardware emulation using the Mentor Graphics Veloce emulation system(optional)

# Design description of AMBA AXI4-Lite protocol

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-chip (SoC) designs. The AMBA AXI-Lite protocol is used for communication with simpler control register style interfaces within components.
AXI-4 Lite Protocol, The Advanced eXtensible Interface (AXI), part of the ARM Advanced Microcontroller Bus Architecture is a parallel high-performance, synchronous, high-frequency, multi-master, multi-slave communication interface, mainly designed for on-chip communication.

The AMBA specification defines three AXI4 protocols:
● AXI4: A high performance memory mapped data and address interface. Capable of Burst access to memory mapped devices.
● AXI4-Lite: A subset of AXI, lacking burst access capability. Has a simpler interface than the full AXI4 interface. Main features are that the address is a traditional Address/Data (single address, single data) and supports only 32 or 64 bits data width.
● AXI4-Stream: A fast unidirectional protocol for transferring data from master to slave.

**Five Channels:**
Write Address Channel: Write Address channels carry the control information of the data to be transferred by the master into the slave memory.
Write Data Channel:Write Data channel involves writing the data from master to slave
Write Response Channel:Slave asserts the valid signal once the write completes that was initiated by the master.
Read Address Channel: Read Address channels carry the control information of the data to be transferred between the master and the slave.
Read Data Channel: Read Data channel involves reading the data at a specified address from slave memory by master.

**Handshake:**
In AXI4 Lite protocol, each channel has its own VALID/READY and uses the same for handshake, in order to transfer control and data information. When the data or control information is available, the source asserts the VALID signal to indicate. The destination asserts the READY signal to indicate that it is available to accept the data or control information. The transfer happens only if both the VALID and READY signals are ASSERTED. Source sends the next DATA or de-asserts VALID. Destination de-asserts READY if it is no longer able to accept DATA.

**Design:**

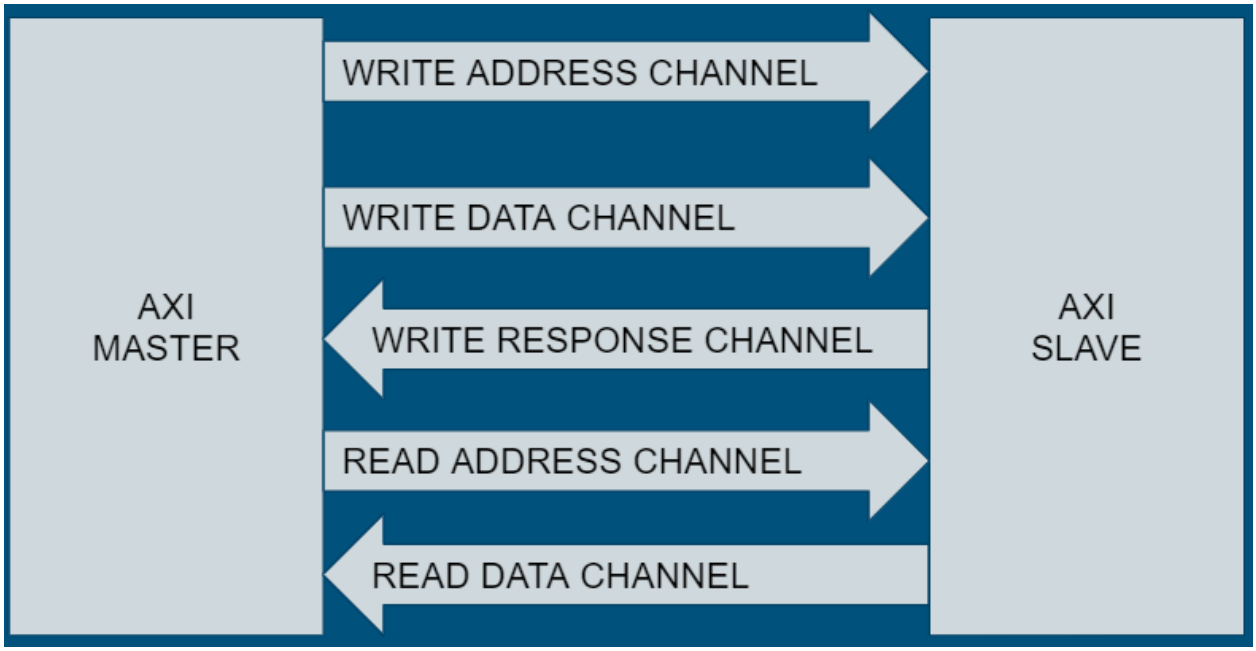
AXI-4 Lite Protocol design includes separate SystemVerilog files for package definition, interface, and modules.

**Package:** In the package module, it defines the parameters including address width and data width of the bus where the address width is considered to be 32 bits. It also includes the enum which defines the four states for read/write operations which is further used in the master module and slave module for the FSM.

**Bus Functional Model:** Similar to Interfaces, BFMs are used for communication between master and slave modules and verification environment. The AXI4 lite BFM module defines all the signals for various channels including Write Address Channel, Write Data Channel, Read Data Channel, Read Address Channel and Write Response Channel. It also includes modport for master and slave to define its signal directions (inout/input/output)

**Master module:** AXI-4 Lite is used for unidirectional transactions. Master module is designed using Finite State Machine, it has two different FSM to read and write data. Master will send signals to slaves to read data from memory (that is declared inside the slave) or to write data to memory in the slave.

**Slave module:** Slave module is designed using two Finite State Machine in which it describes functionality for read transaction and write transaction.



**Write Transaction**

- 1. Master gets the address available on Write Address Channel and gets the data available on Write Data channel, then indicates address and data are valid by asserting AWVALID and WVALID respectively. Master also asserts BREADY to indicate its ready to receive response from the slave.
- 2. Slave will assert AWREADY and WREADY in response on Write Address Channel and Write data Channel respectively,
- 3. READY signals on both Write Address and Data channel handshake happens and then the READY signals can be deasserted.
- 4. Slave will then assert BVALID to indicate a valid response on the Write Response Channel and on the clock transaction is considered done.

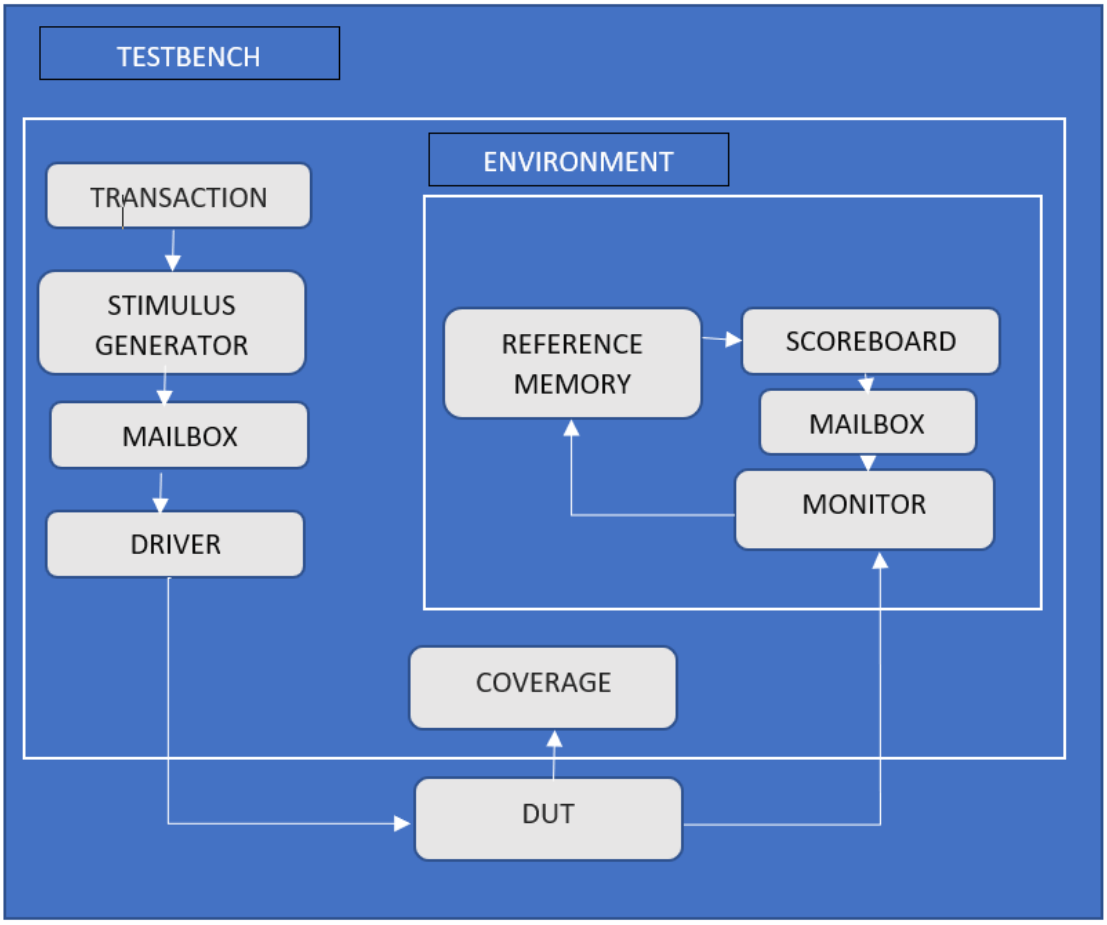
**Read Transaction:**

1. Master gets the address available on Read Address Channel as well as assert ARVALID to indicate that address is valid, and then assert RREADY to indicate master is ready to accept data from slave.
2. After the slave indicates its ready to accept address, the handshake process begins and the slave will place data on the Read Address Channel and assert RVALID to indicate data is valid.
3. Since both RREADY and RVALID is asserted the transaction is consdeired to be completed in next cycle and both the signals will be deasserted further.

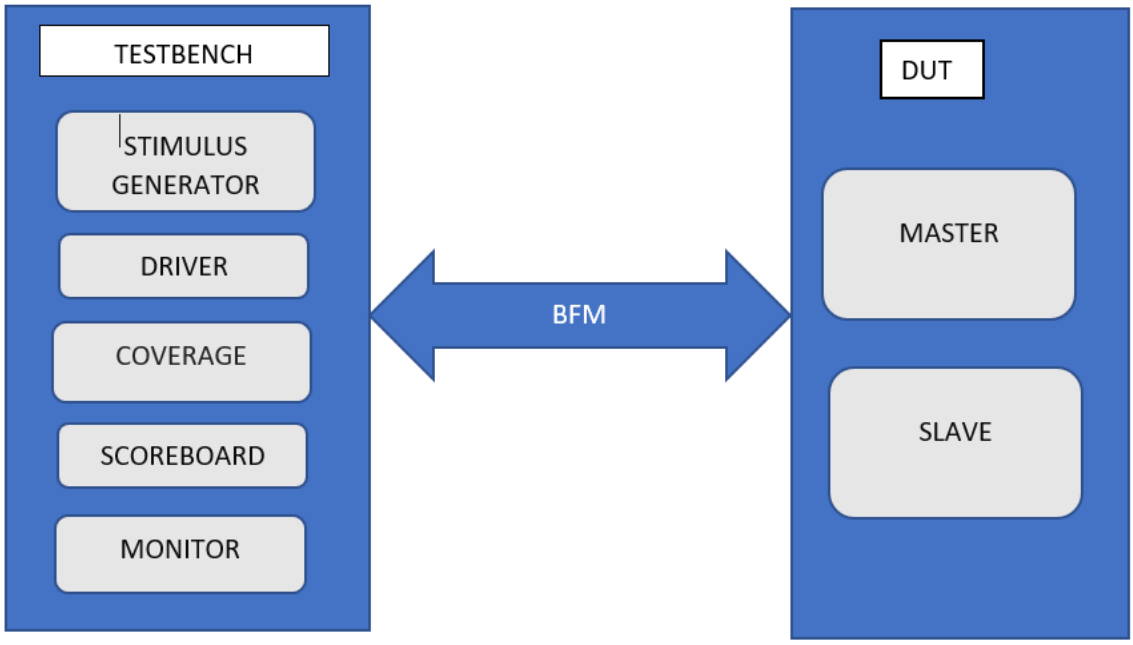
# Extra Material :

**Verification environment:**

**This is an instance for test bench components interactions, just for your reference. It might help you build your verification environment.**



**Interaction between the Design Under Verification and the testbench components**




**Where to submit your deliverables for the project:**

Verification Plan: Submit your Verification Plan to the appropriate dropbox on D2L. Include a .pdf of the Verification Plan in your GitHub repository

Verification Report: Submit your verification report with your final deliverables. The Verification Report should be a completed (test results included) version of your Verification plan. The Verification report should include the coverage statistics that can be provided by QuestaSim, conclusions, lessons learned, next steps, and a work breakdown (which team member did what?)

Demo presentation slides - Include a .pdf of your final project presentation in your final deliverables.

Source code, makefiles, etc.: Include all of the source code you wrote for the project. This code should naturally be in your GitHub repository since you are committing your changes, merging your source code, etc. to GitHub.

**Grading Rubric :**

This project is worth 100 points. There is the possibility of extra credit for projects and project presentations that stand out (in a good way)

25 pts: Verification Plan

40 pts: Final project presentation

25 pts: Quality and contents of your Design Report

10 pts: Quality/readability of your source code (up to)

5 pts: Extra Credit. Extra credit is just that - extra. Your documentation and presentation must be prepared, well written, and well presented for the project to be eligible for extra credit

**Broken model:** We will provide a DUV that we've injected "subtle" bugs into as we get closer to the project due date. This code in this model will be encrypted so that you cannot perform a diff or code review between the working model and the broken model. How do you find the bugs? If you have a good verification strategy and good test cases they should identify the problems. If not, you may need to add additional test cases. It is our intent to give each team their own "broken" model.

## Reference:

Introduction to AXI4-Lite:
https://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081
Specification:
http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf