ECE 593
Fundamentals of Pre-Silicon Validation
Final Project
MESI Intersection Controller
Verification Plan

Monika Sinduja Mullapudi (monikas2@pdx.edu)

Vinitha Baddam (vbaddam@pdx.edu)

Table of Contents

Verification Plan

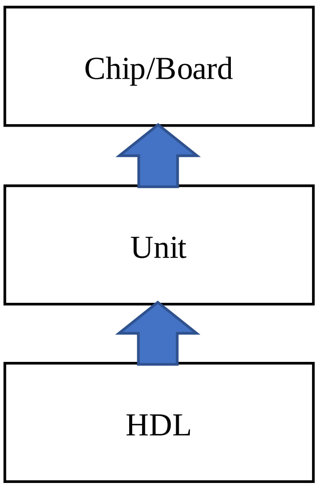
Design Details:

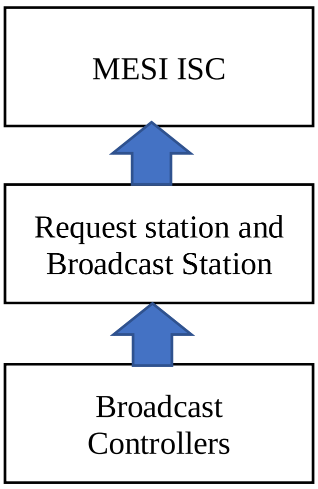
MESI Intersection Controller:

- The MESI Intersection Controller (ISC) is a coherence system controller. It supports the MESI coherence protocol for a cache data consistency. It synchronizes the memory requests of the system masters.
- The controller synchronizes all the caches in the system with the latest copy of data and also the requests from multiple masters. Major elements of coherency mechanism are:
    - ♦ Coherence Controller
    - ♦ Coherency Masters (Different CPUs or Cores) and
    - ♦ Coherency buses.

Description of Verification Levels:

- We can perform verification at top level of DUV hierarchy.
- We do not have any specifications about the unit level or HDL level ports to perform verification at lower levels of hierarchy.
- It would be better to do unit-level verification of the FIFO's and their controller logics first if the HDL level specifications are available.
- This would lay a good verified foundation to move ahead with the higher level of verification. And then in the higher level do not worry about the lower level logic.
- To perform verification at unit/HDL level, the design team need to include the specifications of all the internal modules in the documentation
- The design consists of HDL modules integrated together to form units.





Functions to be Verified:

The design is fairly straightforward in the sense that it implements the MESI protocol and the protocol is defined by coherency action during events. In coherency system like the one we're working with, the masters have to accept information and messages from the coherency controller. The main bus and the coherency bus work together in keeping with the coherence protocol; the main bus transactions are driven by the masters and these masters respond using the main memory, system matrix or the coherency controller. The main transactions/functions we would be accounting for are as follows.

The transactions that are done in the main bus are:

- **Write access:** A write access to the memory (legacy bus transaction).
- **Read access:** A read access to the memory (legacy bus transaction).
- **Write broadcast:** A write broadcast request. Asks for all other master to evict and invalidate data of the requested address. This transaction type is unique for
- coherency systems.
- **Read broadcast:** A read broadcast request. Asks for all other master to evict modified data of the requested address. This transaction type is unique for
- coherency systems.

The transactions that are done in the coherency bus are:

- **Write snoop**: Master request to write to a requested memory location.
- **Read snoop:** Master request to read to a requested memory location.
- **Enable write:** A respond to a write broadcast which is performed in the main bus. It means that the write to the requested memory location can be done.
- **Enable read:** A respond to a read broadcast which is performed in the main bus. It means that the read to the requested memory location can be done.
- Coherency operation starts with the master requesting memory access. A broadcast request to the main memory is sent before any memory access by the master. The coherency controller takes requests from all the masters and collects the responds and follows it up by enabling the initiator to perform the memory access. All coherency controller operations are done in the coherency bus.

Specific Tests and Methods:

Some of the specific tests that we hope to cover most definitely involve basic coherency operations:

- Coherency operation for a write miss.
- Coherency operation for a read miss.
- Coherency operation for a write to a shared line.
- Write miss to an invalid location.
- Write miss to a modified location in other master's cache.
- Two parallel write misses to the same location which is invalid.
- Write miss and a parallel read miss to two different addresses.
- For any given pair of caches check the state of each cache line.

	Cache A			
Cache B	**M**	**E**	**S**	**I**
M	X	X	X	V
E	X	X	X	V
S	X	X	V	V
I	V	V	V	V

We hope to come up with more test cases that we may have missed right now.

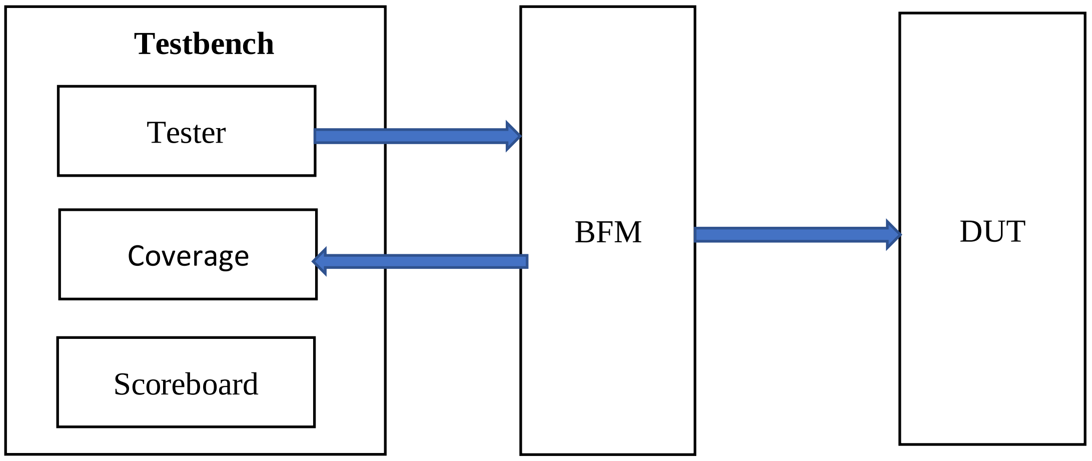
We hope to build an environment that consists of components like scoreboard, tester, coverage block and a bus functional model.

Verification Strategy:

- The drivers, monitors, checkers and scoreboard use only the external interfaces as defined by the specification.
- The internal signals and constructs remain in the black box.

Verification Environment:

Testbench is created with the driver, tester, coverage, scoreboard and monitor components. Stimuli from the tester is provided to the driver and the driver is activated only when the stimulus is present in the FIFO. The stimuli are then provided to the BFM from the driver.



Coverage Requirements:

The coverage goals for the design are based on the functions that we will be verifying. The specific test cases will also help achieve our coverage goals requirements. We'll aim at covering boundary conditions, corner cases, functions, the test cases (and more test cases) and also use constrained randomization to try and achieve a good amount of coverage.

Test Scenarios:

Pervasive functions to be verified for this project:

System reset:

The functions will be verified only if the component can produce the stimulus.

Critical function:

Cache coherency in case of a single master. It will have the main memory and a cache. The following basic functions must be met.

Basic Scenarios:

- The first state of the cache line needs to be invalid at reset and when powered on.
- Check the NOP- behavior of the caches.
- Check the state between multiple memory accesses.

Complex Scenarios:

- Checking coherency in case of a write/read miss.
- Write to or read from an invalid location.

- Checking coherency when write is done in a shared line.
- Write miss to a modified line in other cache.
- Two simultaneous misses to the same address which is invalid.
- Write miss followed by a read miss in different address locations.
- Check the state of the of each cache line depending on the address and instruction.
- Check the transition state in case of read and write operations that can happen randomly.
- Ensuring cache coherency in shared processor that has multiple masters with multiple caches. These masters will generate different addresses.
- Check for data inconsistency scenarios between multiple memory accesses.

Required Tools:

For verification we would require:

1) Software simulation engine - Questasim
2) Waveform Viewer
3) Testbench Components

The testbench environment would communicate with the design through the simulation engine by driving the inputs with a stimulus. The waveform viewer would enable us to observe the behaviour of the design and perform timing analysis. The scoreboard, coverage, monitor and all other testbench components would record the outputs of the design for the corresponding inputs and verify that the behaviour is as expected using various verification strategies.

Risks and Dependencies:

The verification is entirely dependent on the specifications provided. Source code taken from opencores.org has no guarantee of fully working design. This might slow verification team if fixing the design is required. If the specification document is incomplete we may miss out to verify certain test cases. One of the major risks is reliance on a separate verification team for pre-verification of the design at lower level of hierarchy. Since we are only verifying at the top level of DUV, the lower level units are taken as pre-verified components. So, any bugs in the FIFO logics or controlling logics would disrupt the schedule and may also be costly to debug during higher levels of verification. Also, all the tools mentioned above must be available at all times. Unavailability of any of the above requirements would decrease the verification productivity. These risks may cause a delay towards completion of the project.

Resource Requirements

For the MESI ISC design, the resource requirements call for two verification engineers. Designing of the BFM, Scoreboard, Tester, and Coverage block will be split between two team members. Component verification and driving stimulus into the DUV will also be split between 2 team members.

Schedule Details

The schedule for this design is fairly straightforward. We have two resources and a prepared verification plan. We will perform basic functionality testing in 2-3 days. We will create test cases for the environment by the end of first week. In second week we will run our DUV to find bugs which are detected while writing testbench. By the end of third week we will create more test cases and check for corner cases and verify that the design is working as expected.

Task Allocation:

Team Member 1: Designing of the BFM and Scoreboard.

Team Member 2: Designing of the Coverage block and Tester. Component verification and driving stimulus into the DUV will be split between 2 team members.