VERIFICATION TEST PLAN – PROJECT CSCE689

Team 1
Agarwal, Sanket Vinod
Agate, Ashlesha Sunil
Kudva, Dhiraj Dinesh
Kumar, Ishan

OVERALL DESCRIPTION OF THE DESIGN

We aim in simulation-based functional verification of a Multicore MESI based cache design. In our case DUT includes the 4 L1 caches, System bus and the L2 cache. Our main aim is to verify the cache coherency protocol of the system.

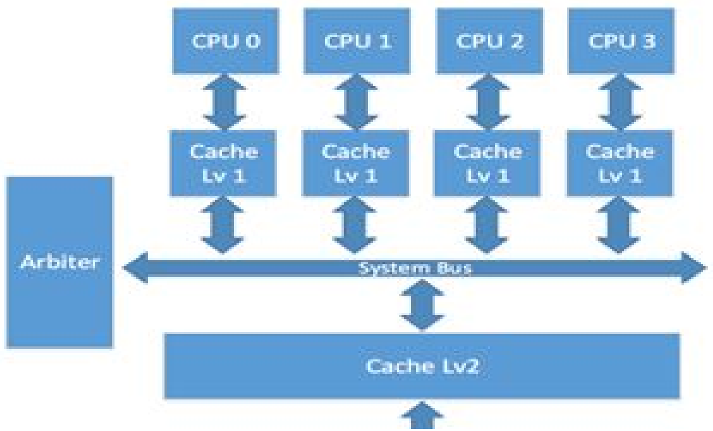
The overall design consists of 4 Level 1 cache that respond to the 4 corresponding CPU cores. There is a system bus that is used as a communication channel between the L1 caches among themselves and L2 cache. Arbiter controls the access grant to the system bus and main Memory is assumed to be a 100% hit i.e. having all the data that can ever be needed. CPU cores send in 32 bit commands (READ/WRITE) to the DUT to which the response is 32 bit as well.

To maintain cache coherency, MESI protocol is followed and pseudo LRU replacement policy used for L1 and L2 caches. Thus, 2 types of FSMs can be expected in the RTL design. Write back and write allocate schemes are used in the caches.

L1 comprises Instruction level and data level cache (address < 32’h4000_000 is Instruction cache and the rest is Data Cache) whereas L2 is a unified cache. L1 is a 256KB each of Instruction level and data level cache with 4-way set associativity whereas L2 is a relatively larger 8MB cache with 8-way set associativity.

The response is synchronous to the positive edge of the clock signal.

Delving deeper into the details we can describe the design in terms of functional blocks



DESCRIPTION OF THE VERIFICATION LEVELS

This seems to be a simple design for which simulation based verification seems to be sufficient. Our system done at one Level of verification -**Sub-system level** is what we choose, which is **flat test-bench based verification**. We can't go further down to block level because we don't have those deeper details. Thus we verify the DUT on a sub-system level which means there is a good compromise between controllability and ease of verification because the design is too big for an IP level verification and SoC level verification would mean less controllability. The design can be further broken down into **IP/block level** to check functionality of each block and ensure that there is no bug in the individual blocks.

Required Tools:

The tools inventory for MESI based cache design verification comprises a single software simulation engine (and license to run it) and one workstation, a waveform viewer, and a test case language or infrastructure. The language or infrastructure must communicate with the simulation engine through the engine's Application Programming Interface (API). The API provides the means to drive the inputs, check the outputs, and clock the model of the design, which is simulated by the engine itself.

Simulation engine software: Cadence 15.20-s022

Features To Be Verified

<u>Global Functions -</u>

1. The time period and the duty cycle should be the same throughout the simulation time.
2. All the functions must occur only at the positive edge of clock
3. Check if all the signals are deasserted or not at the start of any process.

<u>Critical Features –</u>

1. Command-response protocol (32-bits, MAIN func block responds synchronously on positive edge, only one of the instruction level and data level can respond to the processor at a time,)

2. Basic operation of each command(read and write to data level cache (with hits and misses handled effectively), read from instruction level cache(hits and misses handled))

6. No response should be produced if illegal command is issued.(like if write command issued to address <32’h4000_000.) or greater than the memory capacity.

7. Caches should have a size of 256kb only.

8. L1 cache to have 4-way associativity

9. L2 cache to have 8-way associativity

10. Executing write operations on instruction cache. (It should only support read instructions)

11. Pseudo LRU policy test of caches

12. Cache coherency of data cache only.

13. Verify write-back policy i.e. each cache line is not written back to memory every time it is modified.

14.Verify write allocate policy

16. Following the below protocols and schemes- for e.g. send write to one followed by read from another proc.

   a. Cache coherency protocol – MESI (for L1 data level and L2)to be followed otherwise wrong response to commands.

   b. Pseudo-LRU policy for cache replacement

   c. **write back** and **write allocate**

<u>FOR LEVEL 2 CACHE:</u>

1. Cache should be 8 MB
2. Cache is shared across all four cores
3. It is not split into instruction and data cache
4. It has 8 way associativity
5. Pseudo LRU policy
6. No snoop side signals in level 2 cache
7. Write back policy
8. Write allocate policy

<u>Secondary Features –</u>

Some Corner cases included here

1. Send command to instruction and data level simultaneously.

2. more than one CPU cores send commands to the system seeking response.

3. Don't send a grant signal to the system bus and check for output.{System bus works only after grant access signal from the arbiter (that takes the arbiter delay into consideration).}

4. Data_bus_lv1_lv2 should be h'zzzzzzzz when there is no read write operation going and data_in_bus_lv1_lv2 should be low.

<u>Features not to be verified -</u>

1. Arbiter delay and memory delay

2. Internal processes of system bus.//white box verification not to be done

3. Not provided with the hit rate, miss rate data so cannot constrain our random test cases.

4. System bus interface protocols are not provided so cant verify system bus on that level.(to be treated like a black box)

Test Environment: Specific Tests and Methods

White Box : L1 and L2 cache verification.

Grey Box: FSM verification of MESI and pseudo-LRU

Black Box : Arbiter(adding assertin/checkers only) and System bus verification (assertions added)

We can apply checkers which are not directly tested but a part of testbench, in the form of assertions.

For the testing method, a gradual approach can be taken from deterministic to fully random. Starting with pre-generated Directed tests, then on-the-fly directed tests, moving on to pre-generated random tests and finally on-the-fly fully random testing.

Test Scenarios: Matrix (connected back to Features to be Verified)

LEVEL-1 CACHE: (Features Verified : 1,2,6,7,10,11,12,13,14)

1. Test to send multiple reads with variable timing between commands from same cache

Case a: Processor hit, cache line is available in the level 1 cache(all deterministic test cases)

W0-R0-R0-same-addr-data cache
W1-W0-R0-same addr- data cache
W0-W0-R0-same addr-data cache
W0-W1-R1 same addr-data cache
W0-R0-W0-R0 same addr-data-cache

Case b: For a processor miss, there could be a snoop miss or a snoop hit.

b.i) For a snoop miss, we need to access level 2. If there is a miss in level 2 cache then data if fetched from memory. Assuming no replacement

TEST CASES for b.i) all are **DETERMINISTIC**:`
.R0-R0_dcache
.R0-R1_dcache
.R0-R1-W0-same_addr_dcache
.R0-W0-R1-same_addr_dcache
.R0-W1-R0 same addr-dcache
.R0-w1-R1 same addr-dcache
.R0-W0_W0 same addr-dcache
.R0-W0-W1same addr-dcache
.R0-W1-W0 same addr-dcache
.R0-W1-W1 same addr-dcache
.R0-W1-R2 same addr-dcache
.R0-W1-W1 same addr-dcache
.R0-R1-R2 same addr-dcache
.R0-R2-R1 same addr-dcache
.R0-R2-R1-R3-W0-R1 same addr-dcache

b.ii) For a Snoop hit

Test cases are all deterministic

.R0-W1-R0 same addr-dcache (If the cache line is in modified state) here we need to check the access of level 2 being done by processor 1 to write its modified data

.(If the cache line is in shared/ exclusive state, check there is no access to level 2 cache)

. R0-R1 same addr-dcache

.R0-R1-W0 same addr-dcache

.R0-R1-R2 same addr-dcache

.R0-R2-R1 same addr-dcache

.R3-R2-R1-R0 same addr-dcache

Case C: For a miss, there could be a replacement needed if the cache lines are full (Features Verified: 4,11,12)

c.i) if the line to be replaced is in the modified state:

read_replacement _dcache

RAW_replacement_dcache

c.ii) IF the line to be replaced is in shared/exclusive state

Read_only replacement

RAW_replacement_dcache:

4 writes with access_cache_type DCACHE_ACC followed by a read to a different address mapping to the same set from CPU[0]. No other processor should read from the same address. Verify that there is memory writeback.

2. Test to send multiple writes with variable timing between commands from same cache

Case a: For a processor Hit, the same line could be a snoop miss(not present in any other cache) or a snoop hit (present in another cache) (Features Verified: 10,11,12,13)

a.i) For a snoop miss, i.e. only one processor has the line and it is in modified or exclusive state, it can be written directly.

R0-W0-R0 same addr-dcache

R0-W0-R1 same addr-dcache

R0-W0-W0-R0 same addr-dcache

R0-W1-W1 same addr-dcache
W0-W0-R0 same addr-dcache
W0-W0-R1 same addr-dcache
W0-R0-W0 same addr-dcache

a.ii) For snoop hit, i.e. the cache line is in the shared state, others need to be invalidated

R0-R1-W0 same addr-dcache
W0-R1-W0 same addr-dcache
W0-R1-W1same addr-dcache
R1-R0-R2-R3-W0-R1 same addr-dcache

<u>Case b: For processor miss, there could be a snoop miss(not present in any other cache) or a snoop hit (present in another cache) </u>(Features Verified: 13,14,16)

b.i) For snoop miss, level 2 needs to be accessed if miss in level 2 then memory needs to be accessed.

W0-R1 same addr-dcache
W0-W0-R0 same addr-dcache
W0-W0-R1 same addr-dcache
W0-W1-R1 same addr-dcache
W0-W1-R0 same addr-dcache
W0-R0-W0- same addr-dcache
W0-R0-W1 same addr-dcache
W0-R1-W0 same addr-dcache
W0-R1-W1 same addr-dcache
W0-R0-W0 same data same addr-dcache

b.ii) For snoop hit, If it is modified state, it needs to be written back to level 2 and then the processor can write

W0-W1-R0 same addr-dcache
W0-W1-R1 same addr-dcache
R0-W1-W0 same addr-dcache
W0-R0-W1 same addr-dcache
W0-W1-R2 same addr-dcache

b.iii) For snoop hit, if it is in shared state, it needs to be invalidated in other states
R0-R1-W2 same addr-dcache

b.iv) For snoop hit, if it is in the exclusive state, it needs to be in invalidated
R0-W1-R0 same addr-dcache

<u>Case C: Multiple writes to the same address (set in cache : which is modified/exclusive state); shouldn't go to memory. </u>(Features Verified : 2,3,4)

3. Test to check if signals are invalidated (after modifying the value in another processor) [transition from shared to modified] and whether the value from these invalidated. (signals to be checked all_invalidation_done, shared_local)

4. Processor write miss (with free block available with shared/exclusive/modified state): Test to check if the copies in another cache are invalidated before data_in_bus_lv1_lv2 is made high.

5. Read - from maximum and minimum indexed address values.

6. Write to one processor followed with read by another processor - Randomized

7. Write miss to all 4 processors- Randomized

8. Fully random read and write.

9. Write to I-CACHE - Randomized addresses. All should generate errors

10. Read commands to ICACHE and DCACHE of the same processor simultaneously. - only one should respond and then after some delay another one should respond.

11. Send Read/Write commands accessing the same address from 2 or more processors at the same time.(Check whether the arbiter follows the same protocol for such requests).

12. Read and write commands from to DCACHE at the same time.

CHECKING -

We will be checking the results obtained from the DUT with the predetermined expected data values of deterministic test cases.

DESIGN FUNCTIONALITY COVERAGE

We would be using Covergroups and Coverpoints to ensure 100% coverage while carrying out volume verification (randomized test cases). The cases that are not desirable would be marked as illegal bins.

Functional Coverage
Our test plan starts with testing the functionality of cache 0 with different possibilities that can occur like Read Miss, Read Hit, Write Miss, Write Hit. We have also covered situations of MESI protocol during these 4 events. e.g.. Read/Write miss when the snooping line is in Modified/Shared/Exclusive state. This helps in verifying the MESI protocol. In addition, our test cases also cover the situation where a free line is present/not present in cache thereby verifying the write allocate and write back policy of caches.

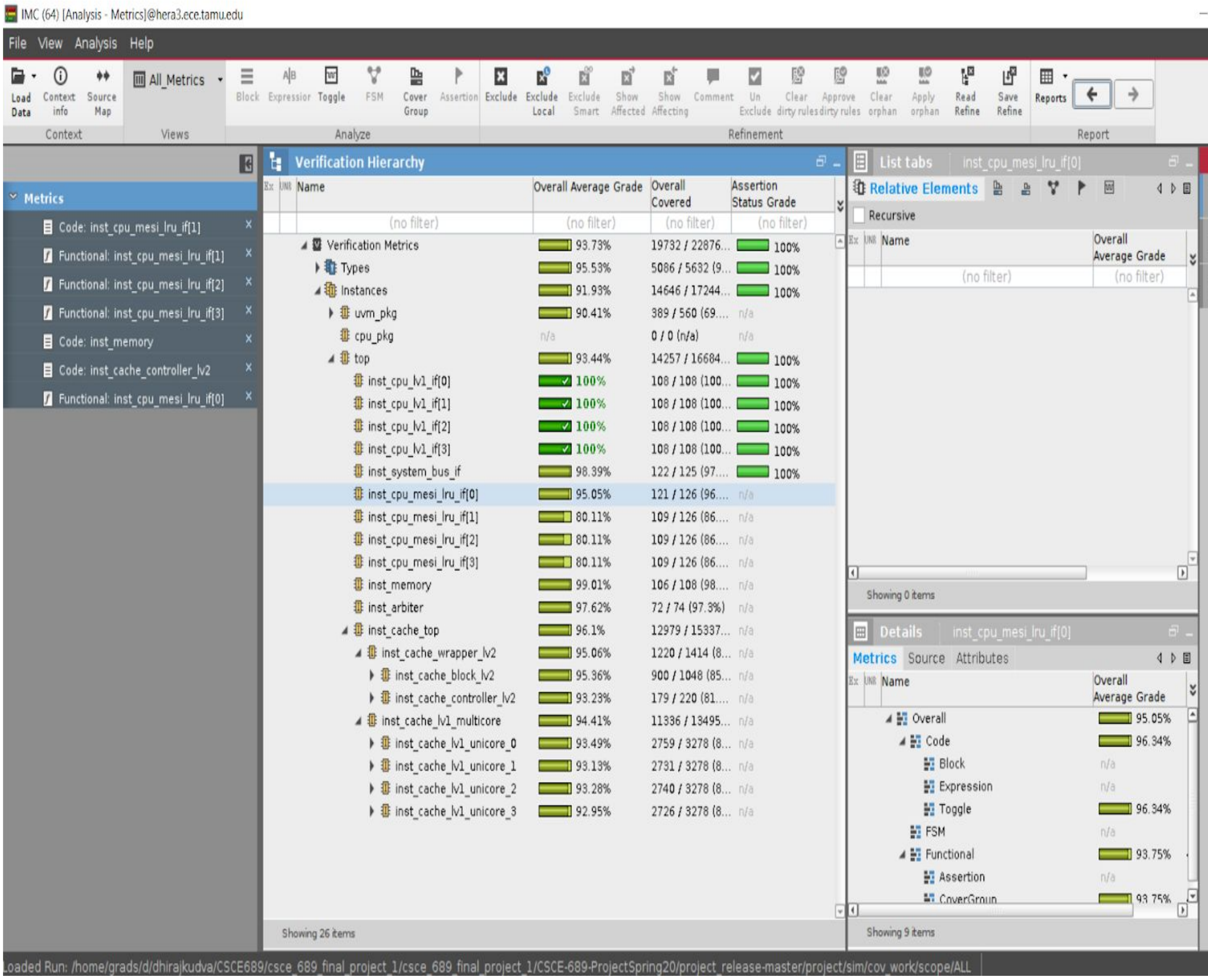
Once Cache 0 is tested deterministically we will be running the same tests with randomized cache and data values i.e. passing values that are generated during simulation, this will help us verify all the caches.

Code Coverage Checkpoints:
Conditional coverage
Loop constructs
FSM code coverage
In our test plan, test cases are written to check whether the design meets the specification such as size and associativity of caches.

Coverage Report Analysis:

The coverage percentage without lru_mesi interface is : **98%**

The coverage percentage with lr_mesi interface is : **95.05%**



IMC (64) [Analysis - Metrics]@hera3.ece.tamu.edu

Metrics

- Code: inst_cpu_mesi_lru_if[1]
- Functional: inst_cpu_mesi_lru_if[1]
- Functional: inst_cpu_mesi_lru_if[2]
- Functional: inst_cpu_mesi_lru_if[3]
- Code: inst_memory
- Code: inst_cache_controller_lv2
- Functional: inst_cpu_mesi_lru_if[0]

Verification Hierarchy

Name	Overall Average Grade	Overall Covered	Assertion Status Grade
(no filter)	(no filter)	(no filter)	(no filter)
Verification Metrics	93.73%	19732 / 22876...	100%
Types	95.53%	5086 / 5632 (9...	100%
Instances	91.93%	14646 / 17244...	100%
uvm_pkg	90.41%	389 / 560 (69....	n/a
cpu_pkg	n/a	0 / 0 (n/a)	n/a
top	93.44%	14257 / 16684...	100%
inst_cpu_lv1_if[0]	✓ 100%	108 / 108 (100...	100%
inst_cpu_lv1_if[1]	✓ 100%	108 / 108 (100...	100%
inst_cpu_lv1_if[2]	✓ 100%	108 / 108 (100...	100%
inst_cpu_lv1_if[3]	✓ 100%	108 / 108 (100...	100%
inst_system_bus_if	98.39%	122 / 125 (97....	100%
inst_cpu_mesi_lru_if[0]	95.05%	121 / 126 (96....	n/a
inst_cpu_mesi_lru_if[1]	80.11%	109 / 126 (86....	n/a
inst_cpu_mesi_lru_if[2]	80.11%	109 / 126 (86....	n/a
inst_cpu_mesi_lru_if[3]	80.11%	109 / 126 (86....	n/a
inst_memory	99.01%	106 / 108 (98....	n/a
inst_arbiter	97.62%	72 / 74 (97.3%)	n/a
inst_cache_top	96.1%	12979 / 15337...	n/a
inst_cache_wrapper_lv2	95.06%	1220 / 1414 (8...	n/a
inst_cache_block_lv2	95.36%	900 / 1048 (85...	n/a
inst_cache_controller_lv2	93.23%	179 / 220 (81....	n/a
inst_cache_lv1_multicore	94.41%	11336 / 13495...	n/a
inst_cache_lv1_unicore_0	93.49%	2759 / 3278 (8...	n/a
inst_cache_lv1_unicore_1	93.13%	2731 / 3278 (8...	n/a
inst_cache_lv1_unicore_2	93.28%	2740 / 3278 (8...	n/a
inst_cache_lv1_unicore_3	92.95%	2726 / 3278 (8...	n/a

Showing 26 items

List tabs inst_cpu_mesi_lru_if[0]

Relative Elements

Recursive

Name	Overall Average Grade
(no filter)	(no filter)

Showing 0 items

Details inst_cpu_mesi_lru_if[0]

Metrics Source Attributes

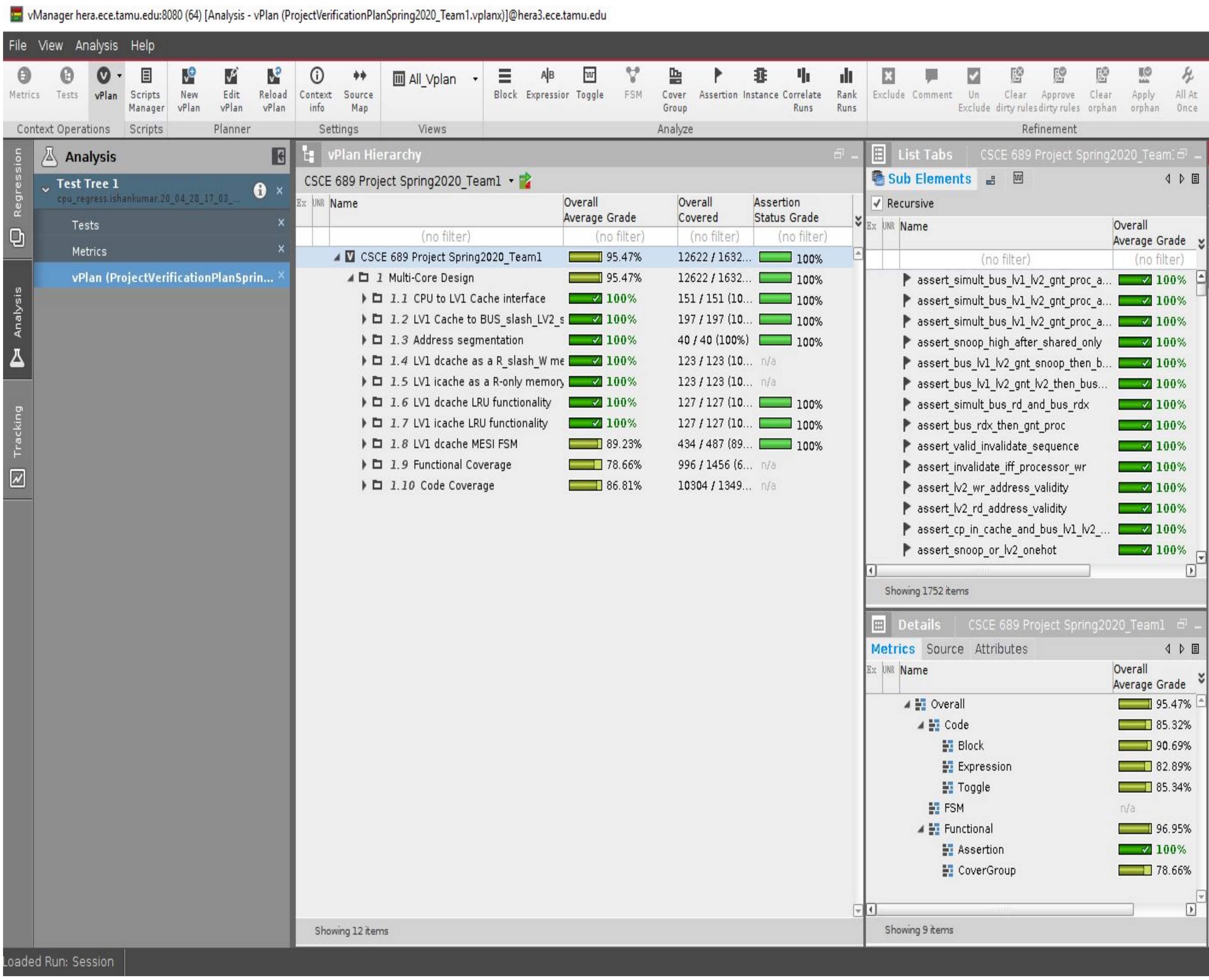
Name	Overall Average Grade
Overall	95.05%
Code	96.34%
Block	n/a
Expression	n/a
Toggle	96.34%
FSM	n/a
Functional	93.75%
Assertion	n/a
CoverGroup	93.75%

Showing 9 items

Loaded Run: /home/grads/d/dhirajkudva/CSCE689/csce_689_final_project_1/csce_689_final_project_1/CSCE-689-ProjectSpring20/project_release-master/project/sim/cov_work/scope/ALL

Coverage Report:

Vmanager Report:



vManager hera.ece.tamu.edu:8080 (64) [Analysis - vPlan (ProjectVerificationPlanSpring2020_Team1.vplanx)]@hera3.ece.tamu.edu

File View Analysis Help
Metrics Tests vPlan Scripts Manager New vPlan Edit vPlan Reload vPlan Context info Source Map All_Vplan Block Expression Toggle FSM Cover Group Assertion Instance Correlate Runs Rank Runs Exclude Comment Un Exclude Clear dirty rules Approve dirty rules Clear orphan Apply orphan All At Once
Context Operations Scripts Planner Settings Views Analyze Refinement
Regression Analysis Tracking
Analysis
Test Tree 1
cpu_regress.ishankumar.20_04_28_17_03_...
Tests
Metrics
vPlan (ProjectVerificationPlanSprin...
vPlan Hierarchy
List Tabs
Sub Elements
Details
Metrics Source Attributes
Loaded Run: Session

CSCE 689 Project Spring2020_Team1

Name	Overall Average Grade	Overall Covered	Assertion Status Grade
(no filter)	(no filter)	(no filter)	(no filter)
CSCE 689 Project Spring2020_Team1	95.47%	12622 / 1632...	100%
1 Multi-Core Design	95.47%	12622 / 1632...	100%
1.1 CPU to LV1 Cache interface	✓ 100%	151 / 151 (10...	100%
1.2 LV1 Cache to BUS_slash_LV2_s	✓ 100%	197 / 197 (10...	100%
1.3 Address segmentation	✓ 100%	40 / 40 (100%)	100%
1.4 LV1 dcache as a R_slash_W me	✓ 100%	123 / 123 (10...	n/a
1.5 LV1 icache as a R-only memory	✓ 100%	123 / 123 (10...	n/a
1.6 LV1 dcache LRU functionality	✓ 100%	127 / 127 (10...	100%
1.7 LV1 icache LRU functionality	✓ 100%	127 / 127 (10...	100%
1.8 LV1 dcache MESI FSM	89.23%	434 / 487 (89...	100%
1.9 Functional Coverage	78.66%	996 / 1456 (6...	n/a
1.10 Code Coverage	86.81%	10304 / 1349...	n/a

Showing 12 items

List Tabs CSCE 689 Project Spring2020_Team1

Sub Elements

✓ Recursive

Name	Overall Average Grade
(no filter)	(no filter)
assert_simult_bus_lv1_lv2_gnt_proc_a...	✓ 100%
assert_simult_bus_lv1_lv2_gnt_proc_a...	✓ 100%
assert_simult_bus_lv1_lv2_gnt_proc_a...	✓ 100%
assert_snoop_high_after_shared_only	✓ 100%
assert_bus_lv1_lv2_gnt_snoop_then_b...	✓ 100%
assert_bus_lv1_lv2_gnt_lv2_then_bus...	✓ 100%
assert_simult_bus_rd_and_bus_rdx	✓ 100%
assert_bus_rdx_then_gnt_proc	✓ 100%
assert_valid_invalidate_sequence	✓ 100%
assert_invalidate_iff_processor_wr	✓ 100%
assert_lv2_wr_address_validity	✓ 100%
assert_lv2_rd_address_validity	✓ 100%
assert_cp_in_cache_and_bus_lv1_lv2_...	✓ 100%
assert_snoop_or_lv2_onehot	✓ 100%

Showing 1752 items

Details CSCE 689 Project Spring2020_Team1

Name	Overall Average Grade
Overall	95.47%
Code	85.32%
Block	90.69%
Expression	82.89%
Toggle	85.34%
FSM	n/a
Functional	96.95%
Assertion	✓ 100%
CoverGroup	78.66%

Showing 9 items