Project Verification Plan

CSCE-714 – Advanced Hardware Design Functional Verification

Team 13

Dinesh Kumar Palani Samy UIN: 434006323

Sri Hariharan Sriram UIN:534001226

Bhanu Sahithya Vattikuti UIN: 534001989

1) OVERVIEW OF THE DESIGN:

The 32-bit 4 core processor system has the following 10 most important features of the design.

1. It has four L1 caches, a unified L2 cache and a system bus that allows communication between the two levels and an arbiter.
2. The L1 cache is split into instruction level cache(L1-I) and data level cache(L1-D). Each cache in L1 is of the size 256KB and 4- way set associativity.
3. The L2 cache is unified and shared, it does not distinguish between instruction and data cache blocks and is common for all four L1 caches. It is 8MB in size and has an 8-way set associativity.
4. It employs a Physically Indexed and Physically Tagged addressing scheme which means there's
   no Translation lookaside Buffer for virtual to physical address conversion.
5. The main Memory is a stub that was intended to serve all memory requests.
6. The cache coherency protocol employed in L1 cache is MESI Protocol.
7. Pseudo LRU replacement policy is used in both L1 and L2 caches.
8. It employs write-back and write-allocate policy for write requests and doesn't have any write buffers.
9. For both L1 and L2 Read and Write requests, the arbiter is in charge of "grants" and "access" requests.

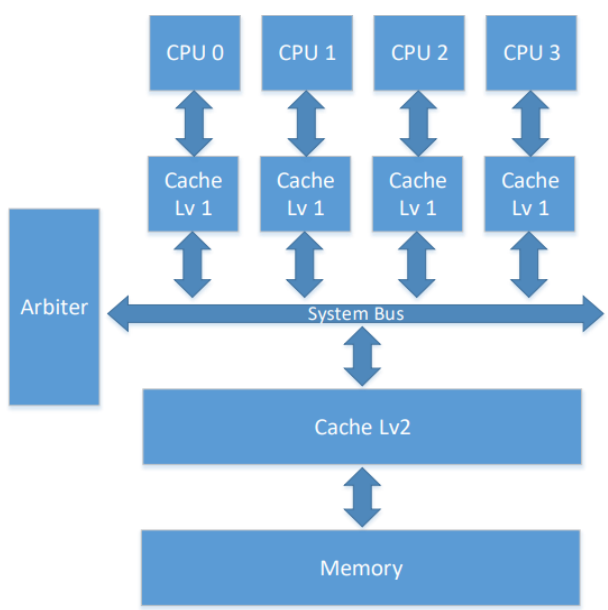


Fig 1: BLOCK DIAGRAM OF DESIGN

2) VERIFICATION LEVELS:

Level of design hierarchy: The module will be verified at a sub-system level since it is an integration of various IP's (L1 cache, L2 cache, system bus).
We followed a grey box verification approach since we have access to various internal signals like the signals between L1 and L2 cache. The functionality of the design is clearly documented in the HAS specification.

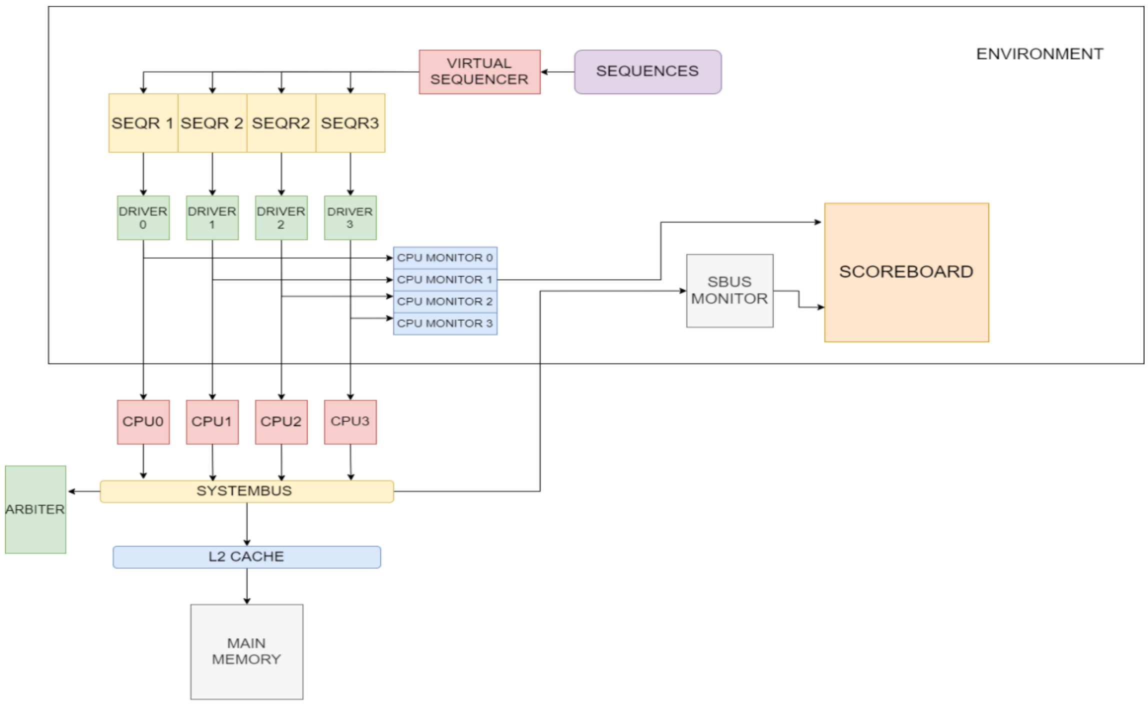


Fig 2: UVM Testbench hierarchy

3) FEATURES:

1. Any miss in L1 data and instruction caches will be served by L2 cache if none of the L1 caches have that data.
2. Replacement policy: The replacement policy used is Pseudo-LRU. We will force a block replacement in L1 cache and check if the policy is adhered.
3. 4- Way set associativity: After filling four ways in a set, an issue to a first block will be a hit.
4. MESI Coherence protocol: Coherency should be maintained among the L1 caches according to MESI protocol.
5. Address range should be in physical address space and not in virtual address space.
6. Writes operations should not be done on L1 Instruction cache.
7. L2 cache is unified so it should be able to serve both L1 instruction and data cache.

8 Since it’s a write allocate cache, the write miss will fetch the block to L1 cache. Consecutive writes to the same block will be a hit.
9 Since it is a write-back cache, the dirty block is written back to L2 only when it is replaced.
10 Multiple request handling: If multiple cores try to access the system bus at the same time, only one processor should access the bus according to the arbitration scheme.

4) TEST CASES AND SCENARIOS:

1. L1 CACHE

1.1 MISS SCENARIO

1.1.1 READ MISS (I-cache & D-cache)

SCENARIO: A read miss to the L1 D-cache or L1 I-cache will result in the block being fetched either from L2 cache or from other L1 caches.

Tests:

- read_request_miss
- read_misses
- read_after_read
- reads_cov
- read_miss_icache

FEATURES COVERED:

It covers feature 1,4 and 7. In feature 4, it covers MESI state transitions like INVALID to EXCLUSIVE when the block is not present in L1 cache. Or if the block is present in the snoop side, transition happens from EXCLUSIVE to SHARED or INVALID to SHARED.

1.1.2 WRITE MISS (D-cache)

SCENARIO: A write miss to L1 D-cache will result in the request being served by L2 and invalidating other copies in L1 D-cache.

Tests:

- write_after_write
- write_tests
- write_request_miss

- write_covs

FEATURES COVERED:

It covers feature 1,4, 6 and 8. When a first write request occurs, it will result in a miss and L2 will serve the data block (feature 1 & 8). The MESI state transitions from INVALID to MODIFIED (feature 4). Also, other copies will be invalidated. On the snoop side, the MESI state transitions from either Exclusive, shared, or modified to INVALID (feature 4). A write request to I- cache is an invalid scenario (feature 6)

1.2 HIT SCENARIO

1.2.1 READ HIT (I- cache & D-cache)

TEST CASES:

- read_after_read
- reads_cov
- read_icache
- req_serv_by_cov

FEATURES COVERED:

It covers feature 4 and 8. When a read hit occurs the MESI state remains unchanged(feature 4).When a read follows a write to the same processor , it will result in a hit(feature 8).

1.2.2 WRITE HIT (D-cache)

SCENARIO: When there's a copy of block present in the processor, it will result in a write hit, which might make other copies invalid.

TEST CASES:

- write_tests
- write_after_write
- write_hit_read
- write_covs

FEATURES COVERED:

It covers feature 4 and 8. When a write hit occurs, the MESI transition depends on the initial state of the block. If the block is in SHARED state, then other copies will be invalidated and MESI state changes to MODIFIED (feature 4). Else, the state transition would be from EXCLUSIVE to MODIFIED or the block would stay in MODIFIED (feature 4).

1.3 READ AFTER WRITE/ WRITE AFTER READ

SCENARIO: This involves mixes of multiple reads and writes in the same test cases.

TESTS:

- random_read_write
- random_write_read
- write_read_many
- write_read_dcache
- read_write_dcache
- write_read_icache
- write_after_read
- random_req_type_data
- bus_req_type
- bus_req_proc_num_cov
- bud_req_snoop_cov

FEATURES COVERED:

It covers 1, 4, 6, 7 and 8.

1.4 REPLACEMENT SCENARIO

SCENARIO: This covers cases where a block gets replaced when a read or write miss occurs.

TESTS:

- replacement
- replacement_writes
- replacement_with_5reads
- eviction
- write_miss_replacement
- writeback_read_replacement
- write_replacement_cov

FEATURES COVERED:
It covers features 2, 3, 4, 8 and 9.

5. ASSERTIONS:

5.1 CPU_LV1_INTERFACE:

1. cpu_wr and cpu_rd should not be asserted at the same clock cycle.
2. As long as cpu_rd is high, addr_bus_cpu_lv1 should hold value
3. Once cpu_rd is asserted, the data_in_bus_cpu_lv1 should be asserted eventually.
4. cpu_rd will not be high when data_in_bus_cpu_lv1 asserted

5.2 SYSTEM_BUS_INTERFACE:

1. lv2_wr_done should not be asserted without lv2_wr being asserted in previous cycle.
2. Data_in_bus_lv1_lv2 and cp_in_cache should not be asserted without lv2_rd being asserted in previous cycle.

3. lv2_rd should be asserted before cp_in_cache is asserted.

4. bus_rd and lv2_rd are asserted only after bus_lv1_lv2_gnt_proc is asserted.

5. If bus_rdx and lv2_rd is asserted, then in the previous cycle bus_lv1_lv2_gnt_proc should be asserted.

6. Both bus_lv1_lv2_gnt_proc and bus_lv1_lv2_gnt_snoop are one-hot signals.

7. Only if proc grants are assigned, snoop grants are assigned.

8. If bus_lv1_lv2_req_snoop is asserted then cp_in_cache is also asserted.

9. If bus_lv1_lv2_gnt_proc is asserted, and if in the next cycle addr_bus_lv1_lv2 is asserted and lv2_rd and lv2_wr is not asserted in the same cycle invalidate signal should get asserted.

10. Invalidate should be followed by all_invalidation_done.

11. After bus_lv1_lv2_gnt_snoop is asserted in a particular clock cycle, from the next clock cycle or in the later clock cycles, the signals 'shared' and 'data_in_bus_lv1_lv2' will be asserted at the same time.

12. When signal invalidate is high, signal bus_lv1_lv2_gnt should be asserted.

13. bus_rd and bus_rdx should not be asserted simultaneously.

14. bus_rd and invalidate should not be asserted at the same time.

15. bus_rdx and invalidate should not be asserted at the same time.

16. bus_lv1_lv2_gnt_proc not asserted when the corresponding req was deasserted for cache.

17. addr_bus_lv1_lv2 should be valid when bus_rdx/lv2_rd is asserted.

6. COVERAGE:

6.1. FUNCTIONAL COVERAGE:

CPU_MONITOR coverage:

- Coverpoints for request type, data, address, address type, illegal.
- Cross coverage between request type, address.
- Cross coverage between request type and data.
- Cross coverage between request type, and address type with ignore bins for ignoring write operations to I-cache .

SYSTEM_BUS_MONITOR:

- Coverpoints for bus_request_type, bus_request_proc_num, bus_req_address, read data, write data snoop, bus request snoop, snoop write request flag, request_serviced_by with ignore_bin for SERV_NONE, cp_in_cache, shared, proc_evict_dirty_blk_addr, proc_evcit_dirty_blk_data, proc_evcit_dirty_blk_flag.
- Cross coverage for REQUEST_TYPE and REQUEST_PROCESSOR.
- Cross coverage for REQUEST_ADDRESS and REQUEST_PROCESSOR.
- Cross coverage for REQUEST_PROCESSOR and WR_DATA_SNOOP.
- Cross coverage for REQUEST_PROCESSOR and BUS_REQ_SNOOP.
- Cross coverage for REQUEST_PROCESSOR and REQUEST_SERVICED_BY.

Overall Functional Coverage: 96.39%

6.2 CODE COVERAGE:

- We covered expressions, toggle and block code coverages to ensure that all parts of the code has been tested. We achieved an overall code coverage of 86.6%
- The overall code coverage of 86.6% includes 88.34% of block coverage, 83.58% of expression coverage and 87.87% of toggle coverage.

COVERAGE HOLES:

CPU Lv1 Interface:

We have not included num_cycles in our coverage plan since this is not an important parameter for verifying our design. Also, we have not included “illegal” because we all transactions that we generate are legal transactions and no write to ICACHE (illegal) can happen.

We have achieved 100% coverage for all coverpoints except for the cross coverage between REQUEST_TYPE and ADDRESS. This is because the REQUEST_TYPE cannot be a WRITE_REQ to an address that falls in ICACHE range.

System Bus Interface:

- For BUS_REQUEST_SNOOP we have included an ignore bin for 4’b1111 since this is not a possible value. This is because all four processors cannot request for snoop access at the same time since atleast one of them should be initiating the request.
- For REQ_SERVICED_BY, an ignore bin is used for SERV_NONE is used since it does not signify any CPU in the design and is declared to -1.

- PROC_EVICT_DIRTY_BLK_ADDR might not give 100% coverage since we cannot evict a dirty block from ICACHE.

- The cross coverage between REQ_PROC and REQ_SNOOP will give a low coverage value, the processor requesting the transfer cannot request for snoop access at the same time. For example, if CPU0 initiates request, then all snoop values with lsb bit as 1 will not be hit.

- The cross coverage between REQ_PROC and REQ_SERVICED_BY will not hit four bins as the processor initiating request cannot serve itself. So the missing bins will have values [REQ_PROC0, SERV_SNOOP0], [REQ_PROC1, SERV_SNOOP1], [REQ_PROC2, SERV_SNOOP2], [REQ_PROC3, SERV_SNOOP3].

Vmanager outputs:

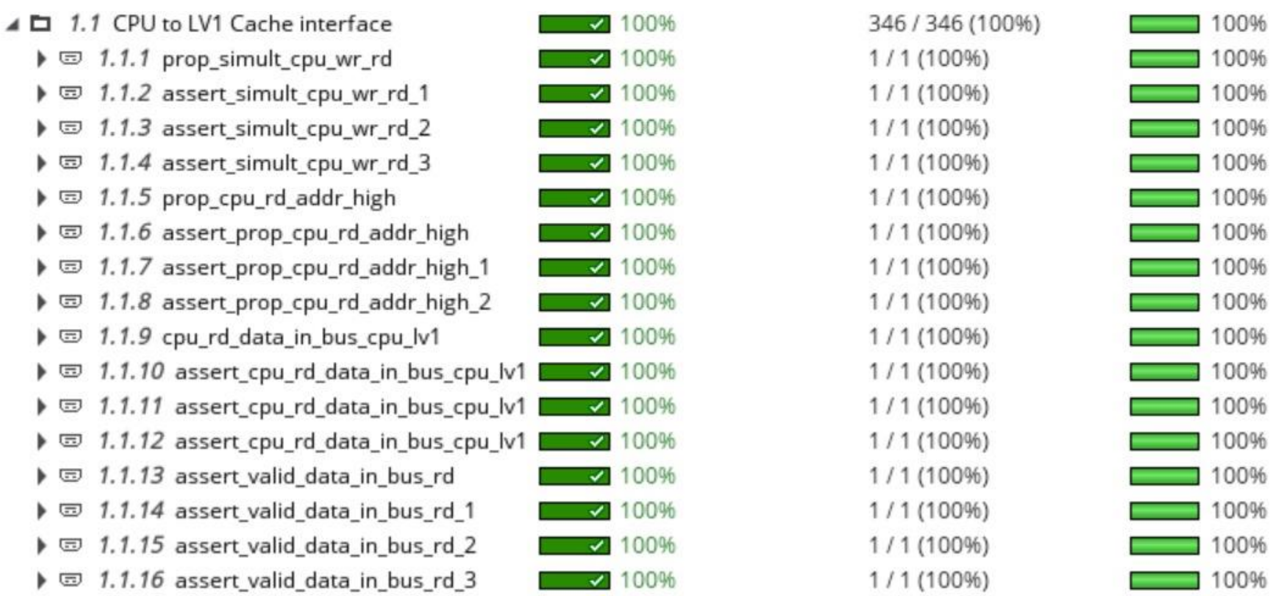


Fig 1: CPU to Level 1 assertions

1.2 LV1 Cache to BUS_slash_LV2_slash_Mem	100%	21 / 21 (100%)	100%
1.2.1 assert_lv2_wr_done	100%	1 / 1 (100%)	100%
1.2.2 assert_lv2_rd_cp_in_cache	100%	1 / 1 (100%)	100%
1.2.3 assert_gnt_proc_bus_rd_lv2_rd	100%	1 / 1 (100%)	100%
1.2.4 assert_gnt_proc_bus_rdx_lv2_rd	100%	1 / 1 (100%)	100%
1.2.5 assert_gnt_proc_onehot	100%	1 / 1 (100%)	100%
1.2.6 assert_gnt_snoop_onehot	100%	1 / 1 (100%)	100%
1.2.7 assert_prop_busrdx_lv2_rd	100%	1 / 1 (100%)	100%
1.2.8 assert_prop_snoop_gnt_after_proc_g	100%	1 / 1 (100%)	100%
1.2.9 assert_snoop_follow_cp_in_cache	100%	1 / 1 (100%)	100%
1.2.10 assert_gnt_addr_invalidate	100%	1 / 1 (100%)	100%
1.2.11 assert_invalidate_follow_all	100%	1 / 1 (100%)	100%
1.2.12 assert_prop_bus_snoop_shared_da	100%	1 / 1 (100%)	100%
1.2.13 assert_invalidate_gnt_proc	100%	1 / 1 (100%)	100%
1.2.14 assert_prop_bus_rd_bus_rdx	100%	1 / 1 (100%)	100%
1.2.15 assert_prop_bus_rd_invalidate	100%	1 / 1 (100%)	100%
1.2.16 assert_prop_bus_rdx_invalidate	100%	1 / 1 (100%)	100%
1.2.17 assert_bus_gnt_deassert_proc_c0	100%	1 / 1 (100%)	100%
1.2.18 assert_bus_gnt_deassert_proc_c1	100%	1 / 1 (100%)	100%
1.2.19 assert_bus_gnt_deassert_proc_c2	100%	1 / 1 (100%)	100%
1.2.20 assert_bus_gnt_deassert_proc_c3	100%	1 / 1 (100%)	100%
1.2.20 assert_bus_gnt_deassert_proc_c3	100%	1 / 1 (100%)	100%
1.2.21 assert_valid_addr_check_lv2_rdx	100%	1 / 1 (100%)	100%

Fig 2: LV1-LV2 assertions

1.1.17 random_read_write	100%	50 / 50 (100%)
1.1.18 random_write_read	100%	10 / 10 (100%)
1.1.19 read after read	100%	10 / 10 (100%)
1.1.20 reads_cov	100%	50 / 50 (100%)
1.1.21 write after read	100%	10 / 10 (100%)
1.1.22 write after write	100%	10 / 10 (100%)
1.1.23 write_read_many	100%	10 / 10 (100%)
1.1.24 write_tests	100%	10 / 10 (100%)
1.1.25 random_request_type	100%	10 / 10 (100%)
1.1.26 bus_req_type	100%	10 / 10 (100%)
1.1.27 bus_req_proc	100%	50 / 50 (100%)
1.1.28 bus_req_snoop	100%	50 / 50 (100%)
1.1.29 req_serv_by	100%	10 / 10 (100%)
1.1.30 cov_snoop_cpu	100%	10 / 10 (100%)
1.1.31 cov_snnop_req_cpu	100%	10 / 10 (100%)
1.1.32 evict_addr_proc	100%	10 / 10 (100%)
1.1.33 cpu_monitor	100%	10 / 10 (100%)

1.3 LV1 dcache as a R_slash_W memory	100%	80 / 80 (100%)	n/a
1.3.1 write_read_dcache	100%	10 / 10 (100%)	n/a
1.3.2 read_misses	100%	10 / 10 (100%)	n/a
1.3.3 write_tests	100%	10 / 10 (100%)	n/a
1.3.4 write_covs	100%	50 / 50 (100%)	n/a
1.4 LV1 icache as a R-only memory	100%	10 / 10 (100%)	n/a
1.4.1 read_miss_icache	100%	10 / 10 (100%)	n/a

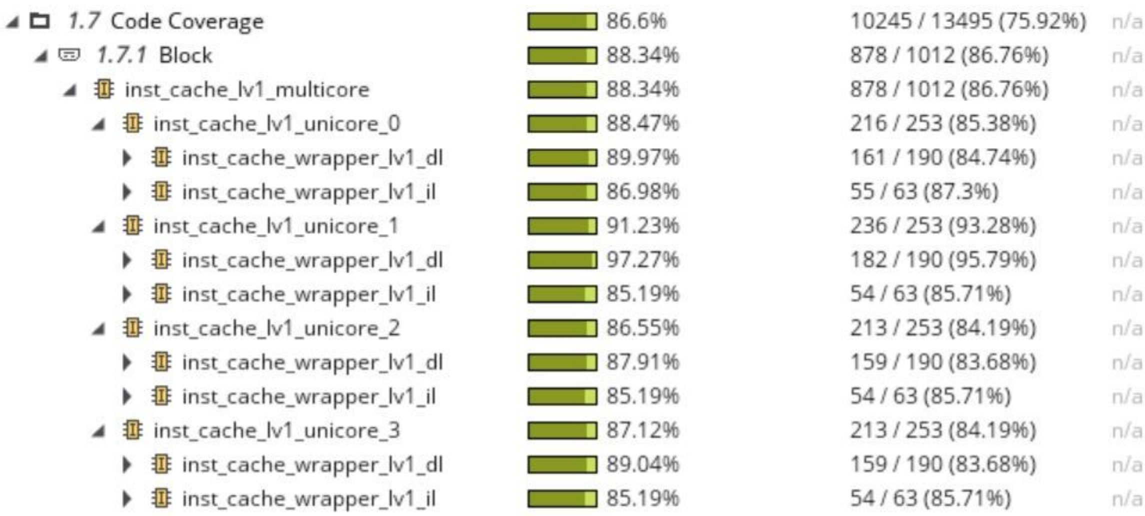
Fig 3: Test cases

1.6 Functional Coverage	96.39%	463 / 510 (90.78%)	n/a
1.6.1 CPU I_slash_O	98.33%	180 / 188 (95.74%)	n/a
cover_cpu_packets	98.33%	45 / 47 (95.74%)	n/a
REQ_TYPE	100%	2 / 2 (100%)	n/a
DATA	100%	10 / 10 (100%)	n/a
ADDRESS	100%	10 / 10 (100%)	n/a
ADDRESS_TYPE	100%	2 / 2 (100%)	n/a
X_REQTYPE__ADDR	90%	18 / 20 (90%)	n/a
X_REQTYPE__ADDRTYPE	100%	3 / 3 (100%)	n/a
cover_cpu_packets	98.33%	45 / 47 (95.74%)	n/a
cover_cpu_packets	98.33%	45 / 47 (95.74%)	n/a
cover_cpu_packets	98.33%	45 / 47 (95.74%)	n/a

Fig 4: CPU-Lv1 functional coverage

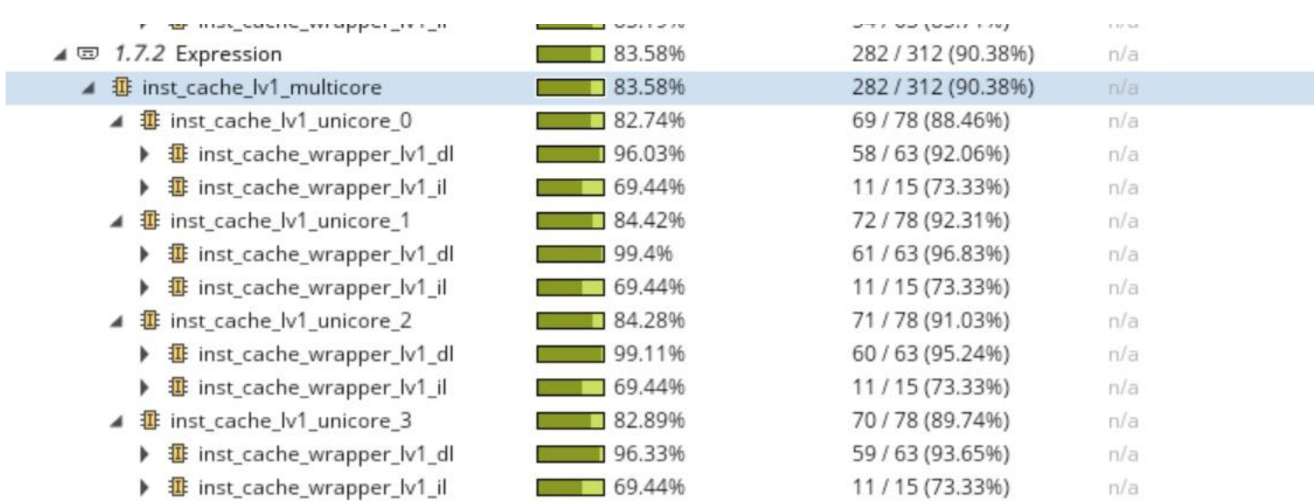
1.6.2 BUS	94.44%	283 / 322 (87.89%)	n/a
cover_system_bus	94.44%	283 / 322 (87.89%)	n/a
REQUEST_TYPE	100%	4 / 4 (100%)	n/a
REQUEST_PROCESSOR	100%	4 / 4 (100%)	n/a
REQUEST_ADDRESS	100%	20 / 20 (100%)	n/a
READ_DATA	100%	20 / 20 (100%)	n/a
WR_DATA_SNOOP	100%	10 / 10 (100%)	n/a
BUS_REQUEST_SNOOP	93.33%	14 / 15 (93.33%)	n/a
SNOOP_WR_REQUEST_FLAG	100%	2 / 2 (100%)	n/a
REQUEST_SERVICED_BY	100%	5 / 5 (100%)	n/a
CP_IN_CACHE	100%	2 / 2 (100%)	n/a
SHARED	100%	2 / 2 (100%)	n/a
PROC_EVICT_DIRTY_BLK_ADDR	80%	8 / 10 (80%)	n/a
PROC_EVICT_DIRTY_BLK_DATA	100%	10 / 10 (100%)	n/a
PROC_EVICT_DIRTY_BLK_FLAG	100%	2 / 2 (100%)	n/a
X_PROC_REQ_TYPE	100%	16 / 16 (100%)	n/a
X_PROC_ADDRESS	100%	80 / 80 (100%)	n/a
X_PROC_WR_DATA	100%	40 / 40 (100%)	n/a
X_PROC_SNOOP	46.67%	28 / 60 (46.67%)	n/a
X_PROC_SERVICED_BY	80%	16 / 20 (80%)	n/a

Fig 5: Lv1-Lv2 functional coverage



1.7 Code Coverage	86.6%	10245 / 13495 (75.92%)	n/a
1.7.1 Block	88.34%	878 / 1012 (86.76%)	n/a
inst_cache_lv1_multicore	88.34%	878 / 1012 (86.76%)	n/a
inst_cache_lv1_unicore_0	88.47%	216 / 253 (85.38%)	n/a
inst_cache_wrapper_lv1_dl	89.97%	161 / 190 (84.74%)	n/a
inst_cache_wrapper_lv1_il	86.98%	55 / 63 (87.3%)	n/a
inst_cache_lv1_unicore_1	91.23%	236 / 253 (93.28%)	n/a
inst_cache_wrapper_lv1_dl	97.27%	182 / 190 (95.79%)	n/a
inst_cache_wrapper_lv1_il	85.19%	54 / 63 (85.71%)	n/a
inst_cache_lv1_unicore_2	86.55%	213 / 253 (84.19%)	n/a
inst_cache_wrapper_lv1_dl	87.91%	159 / 190 (83.68%)	n/a
inst_cache_wrapper_lv1_il	85.19%	54 / 63 (85.71%)	n/a
inst_cache_lv1_unicore_3	87.12%	213 / 253 (84.19%)	n/a
inst_cache_wrapper_lv1_dl	89.04%	159 / 190 (83.68%)	n/a
inst_cache_wrapper_lv1_il	85.19%	54 / 63 (85.71%)	n/a

Fig 6: Code block coverage



1.7.2 Expression	83.58%	282 / 312 (90.38%)	n/a
inst_cache_lv1_multicore	83.58%	282 / 312 (90.38%)	n/a
inst_cache_lv1_unicore_0	82.74%	69 / 78 (88.46%)	n/a
inst_cache_wrapper_lv1_dl	96.03%	58 / 63 (92.06%)	n/a
inst_cache_wrapper_lv1_il	69.44%	11 / 15 (73.33%)	n/a
inst_cache_lv1_unicore_1	84.42%	72 / 78 (92.31%)	n/a
inst_cache_wrapper_lv1_dl	99.4%	61 / 63 (96.83%)	n/a
inst_cache_wrapper_lv1_il	69.44%	11 / 15 (73.33%)	n/a
inst_cache_lv1_unicore_2	84.28%	71 / 78 (91.03%)	n/a
inst_cache_wrapper_lv1_dl	99.11%	60 / 63 (95.24%)	n/a
inst_cache_wrapper_lv1_il	69.44%	11 / 15 (73.33%)	n/a
inst_cache_lv1_unicore_3	82.89%	70 / 78 (89.74%)	n/a
inst_cache_wrapper_lv1_dl	96.33%	59 / 63 (93.65%)	n/a
inst_cache_wrapper_lv1_il	69.44%	11 / 15 (73.33%)	n/a

Fig 7: Code Expression coverage

1.7.3 Toggle	87.87%	9085 / 12171 (74.64%)	n/a
inst_cache_lv1_multicore	87.87%	9085 / 12171 (74.64%)	n/a
inst_cache_lv1_unicore_0	85.06%	2116 / 2947 (71.8%)	n/a
inst_cache_wrapper_lv1_dl	82.09%	1219 / 1672 (72.91%)	n/a
inst_cache_wrapper_lv1_il	76.22%	742 / 1115 (66.55%)	n/a
inst_cache_lv1_unicore_1	87.65%	2425 / 2947 (82.29%)	n/a
inst_cache_wrapper_lv1_dl	92.02%	1537 / 1672 (91.93%)	n/a
inst_cache_wrapper_lv1_il	74.06%	733 / 1115 (65.74%)	n/a
inst_cache_lv1_unicore_2	83.41%	2064 / 2947 (70.04%)	n/a
inst_cache_wrapper_lv1_dl	79.29%	1176 / 1672 (70.33%)	n/a
inst_cache_wrapper_lv1_il	74.06%	733 / 1115 (65.74%)	n/a
inst_cache_lv1_unicore_3	84.04%	2100 / 2947 (71.26%)	n/a
inst_cache_wrapper_lv1_dl	81.19%	1212 / 1672 (72.49%)	n/a
inst_cache_wrapper_lv1_il	74.06%	733 / 1115 (65.74%)	n/a

Fig 8: Code Toggle coverage