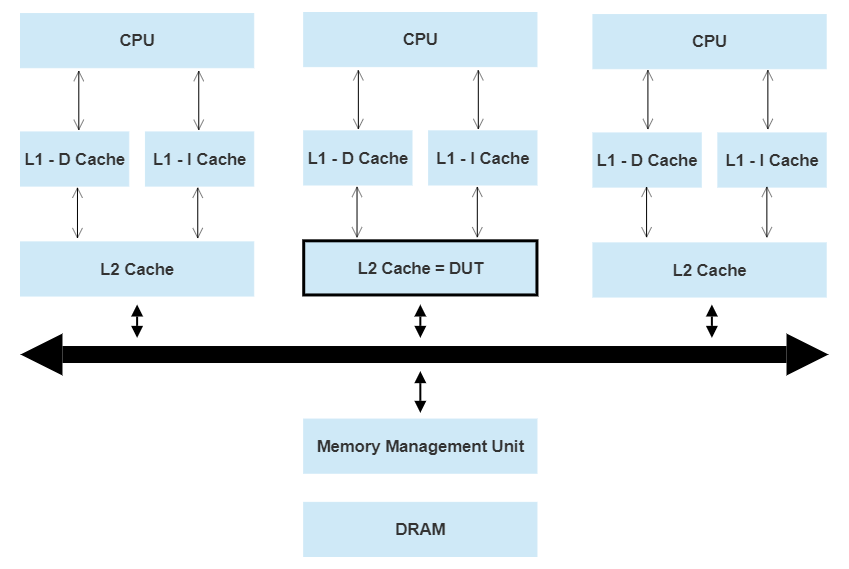
**ECE485/585**

**MICROPROCESSOR SYSTEM DESIGN**

**VERIFICATION PLAN – GROUP 14.**

**Block diagram:**



**Test Plan:**

1. **Interface testing:** This test tests all the interfaces and parameters
   1. Parameters and Ports
   2. Hit counter
   3. Miss counter
   4. Tri-state of MESI bits
2. **Basic memory test:** This test tests the functionality of the cache.
   1. Write to a address.
   2. Read from the same address.
   3. We are not having any data, so it will be a hit since we are reading from the same address it was last written to.
3. **Hit\_miss test:** This test will make a write and read to a random address

Output expected:

* 1. First Miss
  2. Then Hit
  3. a and b will happen the number of times we write and read.

This test will also do an eviction in back but our test is targeted for testing HIT and Miss so we don’t test LRU in this test.

1. **PLRU test:** This test will stress the cache lines/sets by making sure all the decisive statements are exercised during the read and write process.
   1. Write to same set with different tag in random order.
   2. At the 9th Write, check if you are getting the Least Recently Used line evicted.
   3. Also check if eviction is occurring in all lines after 9th write. Use for loop to do it.
   4. Repeat a, b, c for different sets.
2. **MESI FSM test case:**

This will test the following

* 1. If the cache is empty, L1 will be requesting data from L2 and if it is a miss in L2 too, a BusRd signal is initiated to the bus. Upon obtaining data from DRAM, L2 and L1 will change its state from Invalidate to Exclusive. Upon a PRWr to L1(Which we won’t be Modelling), L1 will change its state from Exclusive to Modified. If L1 is evicting the same modified line, there will be a PrWr to L2 and L1 should write back to L2 the evicted line and change its state from Exclusive to Modified.
  2. If the Cache is Empty, it will be in Invalid state. If a PrWr is initiated, L1 will request L2, L2 will do a DRAM read. After receiving, L2 will give to L1. And then Processor will write to L1 and L1 will change its state from Invalid to Modified.
  3. When dealing with two processors, if CPU 1 L2 encounters a PrRd, its state changes from Invalid to Exclusive upon receiving the requested data. A snoop request from CPU 2 L2 for same address will result in CPU 1 L2 Cache change its state from Exclusive to Shared, and asserts C. If there is a BusUpgr/BusRdX from CPU 2 L2, CPU 1 L2 will change its state from Shared to Invalid. PrRd to same address to CPU 1 L2, Cache state changes from Invalid to Shared. If CPU 1 L2 initiates PrWr to the same address, state changes from Shared to Modified, BusUpgr will be asserted high which will make data in L2 of CPU2 as invalid. If CPU 3 L2 initiates a BusRdX, L2 of CPU 1 will go from Modified to Invalid and Flush will be asserted to high therefore DRAM write operation will occur.
  4. When dealing with two processors, if a PrRd is sent to CPU 1 L2, L2 will go from Invalid to Exclusive. CPU 2 L2 will also do a PrRd from the same address as CPU 1 L2. So, Both CPU1 L2 and CPU2 L2 will go to Shared state. Now, there is PrWr in CPU 2 L2, it asserts BusRdX and therefore CPU1 L2 will go to Invalid from Shared and the data in CPU 1 L2 will be stale.
  5. When dealing with two processors, a PrWr will be initiated to CPU 1 L2, L2 will go from Invalid to Modified. CPU 2 L2 will do a PrRd from the same address as CPU 1 L2. So, CPU 1 L2 will go from Modified to Shared after asserting Flush.

1. **Write allocate test/write back:**
   1. We write to a empty cache line. Since its empty, it will be a miss. This will lead to a DRAM read request by the Cache.
   2. The cache line is placed to the cache depending on the PLRU bits and the cache will change its MESI state from “Invalidate” to “Modified”. BusRdX should be asserted high upon this transition.
   3. Repeat a,b for different sets
2. **Inclusivity test**
   1. On a miss to L1 and a hit to L2 /On a miss to L1, hit to L2 but dirty/Modified:

Read/Write request will be initiated. If it is in L2, we will reply by sending the required Cache line. L2 should be written with the address from where the cache line should be sent to L1. On a miss to L1 and a miss to L2, DRAM request will be initiated from L2. DRAM will reply to L2 with data which is asked for and then L2 will reply to L1 by forwarding the same data that arrives at L2. L1 and L2 should make space for the new incoming data from the DRAM.

* 1. On a snopped Upgr/ReadX hit to L2:

L2 should tell L1 invalidate the cache line and also invalidate its own copy.(\*\*Design does not implement a single bit in L2 to indicate “valid” in L1). If L1 does not have the copy which needs to be invalidated, it does not do anything.

* 1. On a snooped ReadX hit to modified line in L2:

L2 requests L1 for the same address. If there are any dirty bits set in L1, then L2 will flush the updated data to DRAM and then invalidate both L1 and L2 copy.

* 1. Repeat a,b,c for different sets

1. **Clear cache test:**
   1. First, make all the Cache states as Invalid
   2. Write to several addresses in the cache
   3. When n is assigned 8, Cache should reset
   4. After c, when we perform a Read/Write to the same addresses, miss counter will be updated to 1 and a DRAM read request will be initiated.